Database Administration and Management

Concurrency Control

When several transactions execute concurrently without any rules and protocols, various problems arise that may harm the data integrity of several databases. These problems are known as concurrency control problems. Therefore several rules are designed, to maintain consistency in the transactions while they are executing concurrently which are known as concurrency control protocols.

A transaction is a single reasonable unit of work that can retrieve or may change the data of a database. Executing each transaction individually increases the waiting time for the other transactions and the overall execution also gets delayed. Hence, to increase the throughput and to reduce the waiting time, transactions are executed concurrently.

Example: Suppose, between two railway stations, **A** and **B**, 5 trains have to travel, if all the trains are set in a row and only one train is allowed to move from station **A** to **B** and others have to wait for the first train to reach its destination then it will take a lot of time for all the trains to travel from station **A** to **B**. To reduce time all the trains should be allowed to move concurrently from station **A** to **B** ensuring no risk of collision between them.

When several transactions execute simultaneously, then there is a risk of violation of the data integrity of several databases. **Concurrency Control in DBMS** is a procedure of managing simultaneous transactions **ensuring their atomicity, isolation, consistency and serializability.**

Concurrent Execution in DBMS

- In a multi-user system, multiple users can access and use the same database at one time, which is known as the concurrent execution of the database. It means that the same database is executed simultaneously on a multi-user system by different users.
- While working on the database transactions, there occurs the requirement of using the database by multiple users for performing different operations, and in that case, concurrent execution of the database is performed.

Concurrency Control Problems

Several problems that arise when numerous transactions execute simultaneously in a random manner are referred to as Concurrency Control Problems.

1. Dirty Read Problem

The dirty read problem in DBMS occurs when a transaction reads the data that has been updated by another transaction that is still uncommitted. It arises due to multiple uncommitted transactions executing simultaneously.

- Example: Consider two transactions A and B performing read/write operations on a data DT in the database DB. The current value of DT is 1000: The following table shows the read/write operations in A and B transactions.

Time	A	B
T1	READ(DT)	------
T2	DT=DT+500	------
T3	WRITE(DT)	------
T4	------	READ(DT)
T5	------	COMMIT
T6	ROLLBACK	------

Transaction A reads the value of data DT as 1000 and modifies it to 1500 which gets stored in the temporary buffer. The transaction B reads the data DT as 1500 and commits it and the value of DT permanently gets changed to 1500 in the database DB. Then some server errors occur in transaction A and it wants to get rollback to its initial value, i.e., 1000 and then the dirty read problem occurs.

Unrepeatable Read Problem

The unrepeatable read problem occurs when two or more different values of the same data are read during the read operations in the same transaction.

Example: Consider two transactions A and B performing read/write operations on a data DT in the database DB. The current value of DT is 1000: The following table shows the read/write operations in A and B transactions.

Time	A	B
T1	READ(DT)	------
T2	------	READ(DT)
T3	DT=DT+500	------
T4	WRITE(DT)	------
T5	------	READ(DT)

Transaction A and B initially read the value of DT as 1000. Transaction A modifies the value of DT from 1000 to 1500 and then again transaction B reads the value and finds it to be 1500. Transaction B finds two different values of DT in its two different read operations.

Lost Update Problem

The Lost Update problem arises when an update in the data is done over another update but by two different transactions.

Example: Consider two transactions A and B performing read/write operations on a data DT in the database DB. The current value of DT is 1000: The following table shows the read/write operations in A and B transactions.

Time	A	B
T1	READ(DT)	------
T2	DT=DT+500	------
T3	WRITE(DT)	------
T4	------	DT=DT+300
T5	------	WRITE(DT)
T6	READ(DT)	------

Transaction A initially reads the value of DT as 1000. Transaction A modifies the value of DT from 1000 to 1500 and then again transaction B modifies the value to 1800. Transaction A again reads DT and finds 1800 in DT and therefore the update done by transaction A has been lost.

Advantages of Concurrency

In general, concurrency means, that more than one transaction can work on a system. The advantages of a concurrent system are:

Database Administration and Management

- **Waiting Time:** It means if a process is in a ready state but still the process does not get the system to get execute is called waiting time. So, concurrency leads to less waiting time.
- **Response Time:** The time wasted in getting the response from the cpu for the first time, is called response time. So, concurrency leads to less Response Time.
- **Resource Utilization:** The amount of Resource utilization in a particular system is called Resource Utilization. Multiple transactions can run parallel in a system. So, concurrency leads to more Resource Utilization.
- **Efficiency:** The amount of output produced in comparison to given input is called efficiency. So, Concurrency leads to more Efficiency.

Disadvantages of Concurrency

- **Overhead:** Implementing concurrency control requires additional overhead, such as acquiring and releasing locks on database objects. This overhead can lead to slower performance and increased resource consumption, particularly in systems with high levels of concurrency.
- **Deadlocks:** Deadlocks can occur when two or more transactions are waiting for each other to release resources, causing a circular dependency that can prevent any of the transactions from completing. Deadlocks can be difficult to detect and resolve, and can result in reduced throughput and increased latency.

Database Administration and Management

- **Reduced concurrency:** Concurrency control can limit the number of users or applications that can access the database simultaneously. This can lead to reduced concurrency and slower performance in systems with high levels of concurrency.
- **Complexity:** Implementing concurrency control can be complex, particularly in distributed systems or in systems with complex transactional logic. This complexity can lead to increased development and maintenance costs.
- **Inconsistency:** In some cases, concurrency control can lead to inconsistencies in the database. For example, a transaction that is rolled back may leave the database in an inconsistent state, or a long-running transaction may cause other transactions to wait for extended periods, leading to data staleness and reduced accuracy.

What is Data?

Data is a collection of a distinct small unit of information. It can be used in a variety of forms like text, numbers, media, bytes, etc. it can be stored in pieces of paper or electronic memory, etc Word 'Data' is originated from the word 'datum' that means 'single piece of information.' It is plural of the word datum. In computing, Data is information that can be translated into a form for efficient movement and processing. Data is interchangeable.

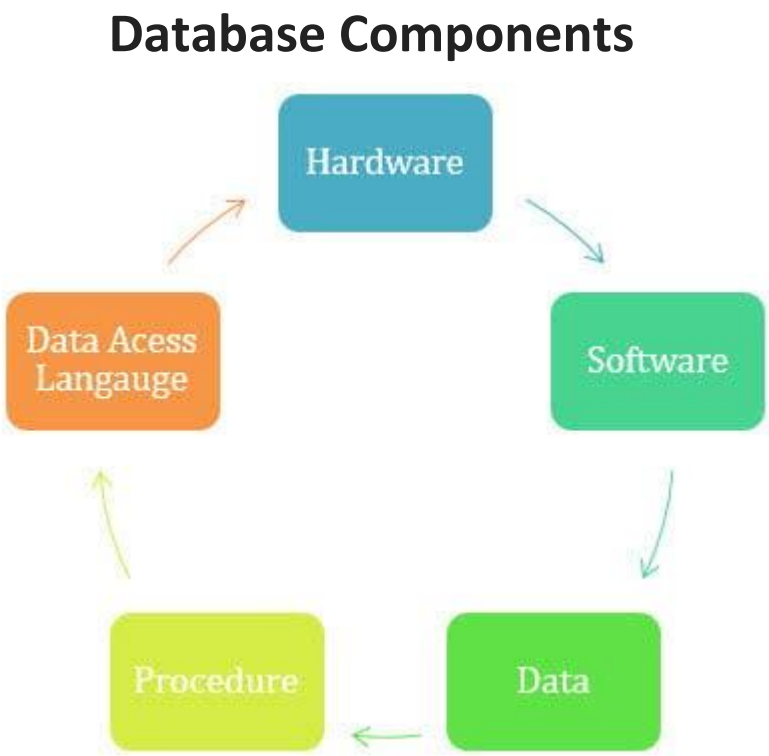
What is Database?

A **database** is an organized collection of data, so that it can be easily accessed and managed .It organize data into tables, rows, columns, and index it to make it easier to find relevant information. **Database handlers** create a database in such a way that only one set of software program provides access of data to all the users. The **main purpose** of the database is to operate a large amount of information by storing, retrieving, and managing data.

There are many **dynamic websites** on the World Wide Web nowadays which are handled through databases. For example, a model that checks the availability of rooms in a hotel. It is an example of a dynamic website that uses a database.

There are many **databases available** like MySQL, Sybase, Oracle, MongoDB, Informix, PostgreSQL, SQL Server, etc.

Modern databases are managed by the database management system (DBMS).**SQL** or Structured Query Language is used to operate on the data stored in a database. SQL depends on relational algebra and tuple relational calculus.



Database Components

There are five main components of a database:

Hardware

The hardware consists of physical, electronic devices like computers, I/O devices, storage devices, etc. This offers the interface between computers and real-world systems.

Software

This is a set of programs used to manage and control the overall database. This includes the database software itself, the Operating System, the network software used to share the data among users, and the application programs for accessing data in the database.

Data

Data is a raw and unorganized fact that is required to be processed to make it meaningful. Data can be simple at the same time unorganized unless it is organized. Generally, data comprises facts, observations, perceptions, numbers, characters, symbols, images, etc.

Procedure

Procedure are a set of instructions and rules that help you to use the DBMS. It is designing and running the database using documented methods, which allows you to guide the users who operate and manage it.

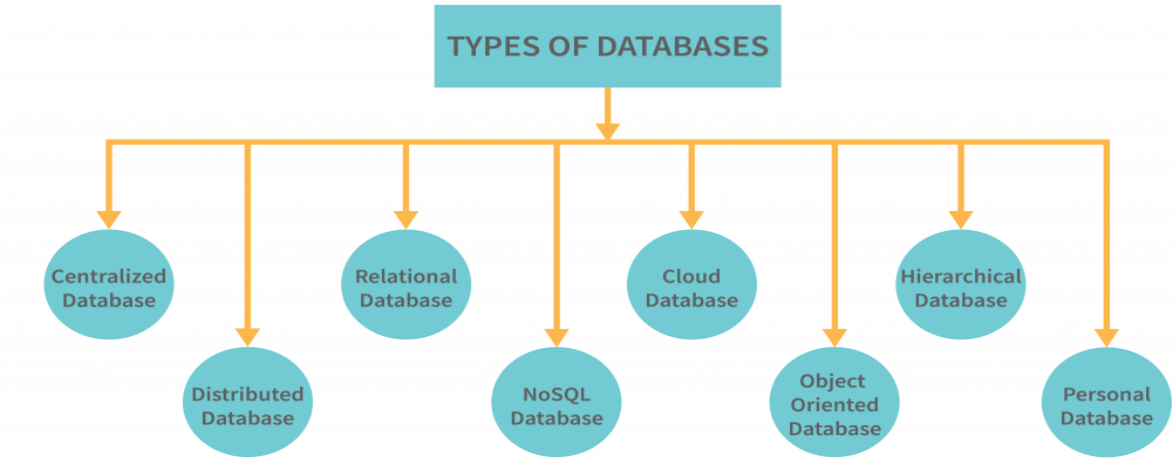
Database Access Language

Database Access language is used to access the data to and from the database, enter new data, update already existing data, or retrieve required data from DBMS. The user writes some specific commands in a database access language and submits these to the database.

Types of Databases

There are different kinds of databases. That are as follow:

.



Centralized Database :

A Database which is located and stored in a single location is called a Centralized Database. The centralized database's location is generally a server CPU or desktop or the mainframe computer which is accessed by the users through a computer network like LAN or WAN. A centralized database (sometimes abbreviated CDB) is a database that is located, stored, and maintained in a single location. This location is most often a central computer or database system, for example a desktop or server CPU, or a mainframe computer.

Distributed Database :

A distributed database is basically a database that is not limited to one system, it is spread over different sites, i.e, on multiple computers or over a network of computers. A distributed database system is located on various sites that don't share physical components. This may be required when a particular database needs to be accessed by various users globally. It needs to be managed such that for the users it looks like one single database. Though there are many distributed databases to choose from, some examples of distributed databases include Apache Ignite, Apache Cassandra, Apache HBase, Couchbase Server, Amazon SimpleDB, Clusterpoint, and FoundationDB.

Relational database:

A relational database is a type of database that stores data in tables made up of rows and columns. Each table represents a specific object in the database like users, products, orders, and so on. Popular examples of standard relational databases include Microsoft SQL Server, Oracle Database, MySQL and IBM DB2.

NoSQL Database:

The term NoSQL, short for "not only SQL," refers to non-relational types of databases. NoSQL databases use a non-tabular format to store data rather than in rule-based, relational tables like relational databases. NoSQL databases use a

flexible schema model that supports a wide variety of unstructured data, such as documents, key-value, wide columns, graphs, and more. Organizations typically choose NoSQL databases for their large amounts of data that don't fit into a relational model, dynamic schema enabling more flexibility, horizontal scalability, and ease of development.

Cloud Database:

A cloud database is a database that is deployed, delivered, and accessed in the cloud. Cloud databases organize and store structured, unstructured, and semi-structured data just like traditional on-premises databases. However, they also provide many of the same benefits of cloud computing, including speed, scalability, agility, and reduced costs.

Object oriented Database:

An object-oriented database is based on the principles of object-oriented programming (OOP). In an object-oriented database, data is organized and stored as objects. Objects are self-contained units that contain both data and the operations or methods that can be performed on that data. It helps in efficient representation and management of complex data structures and relationships. one example of an object-oriented database: ZODB, which is used with Python.

Hierarchical Database:

A hierarchical database is a data model in which data is stored in the form of records and organized into a tree-like structure, or parent-child structure, in which one parent node can have many child nodes connected through links. The IBM Information Management System (IMS) and RDM Mobile are examples of a hierarchical database system with multiple hierarchies over the same data.

Personal Database:

A personal database is one that is designed for a single person. It is typically stored on a personal computer and has a very simple design, consisting of only a few tables. Personal databases are not typically suitable for complex operations, large amounts of data or business operations. Microsoft Access and Open Office Base are examples of personal database-management systems.

Data Independence

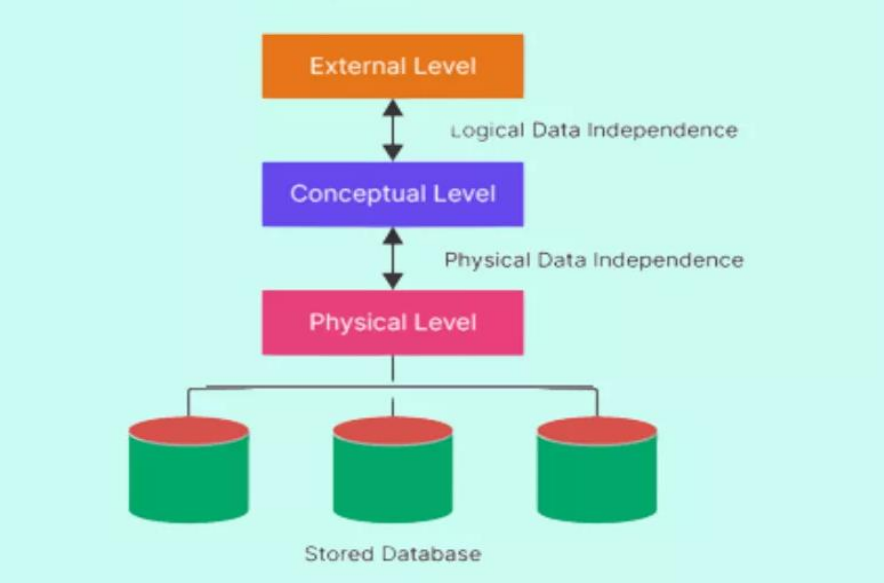
Data Independence means the ability of the data to change the schema at one level of the database without having to change the schema at the next higher level. In simple words, we can say that Data independence is a property of a database that allows the User or Database Administrator to change the schema at one level without affecting the data or schema at another level.

Purpose: The purpose of data independence is to enhance the security of the system, save time and reduce costs needed once the information is changed or altered.

Types of Data Independence

In DBMS there are two types of data independence

1. Physical data independence
2. Logical data independence.



Logical Data Independence

- Logical data independence refers characteristic of being able to change the conceptual schema without having to change the external schema.
- Logical data independence is used to separate the external level from the conceptual view.
- If we do any changes in the conceptual view of the data, then the user view of the data would not be affected.
- Logical data independence occurs at the user interface level.

Physical Data Independence

- Physical data independence can be defined as the capacity to change the internal schema without having to change the conceptual schema.
- If we do any changes in the storage size of the database system server, then the Conceptual structure of the database will not be affected.
- Physical data independence is used to separate conceptual levels from the internal levels.
- Physical data independence occurs at the logical interface level.

Relational Data Model

Data Model

A Data Model in Database Management System (DBMS) is the concept of tools that are developed to summarize the description of the database. Data Models provide us with a transparent picture of data which helps us in creating an actual

database. It shows us from the design of the data to its proper implementation of data.

Relational Data Model

Relational data model is the primary data model, which is used widely around the world for data storage and processing. This model is simple and it has all the properties and capabilities required to process data with storage efficiency.

Concepts

Relational database concepts to understand in DBMS include:

- **Attribute:** An **attribute** is a characteristic or quality of an entity. In the context of the relational model, an attribute is a column in a table.
- **Tables:** A table is a data collection organized into rows and columns. In the relational model for database **management**, tables are used to store information about entities.
- **Tuple:** A **tuple** is a row in a table. Tuples are composed of attributes.
- **Relation Schema:** A relation schema is a blueprint for a table. It defines the attributes that are contained in a table as well as the relationships between those attributes.
- **Degree:** The degree is the number of attributes that it contains.
- **Cardinality:** Cardinality defines the relationship between two attributes in a relation schema. There are three possible relationships: one-to-one, one-to-many, and many-to-many.

- **Column:** A column is an attribute that is contained in a table. Columns are also sometimes referred to as fields.
- **Relation instance:** A relation instance is a set of tuples that conforms to a given relation schema. In other words, it is populated data that has been organized into rows and columns according to the blueprint set forth by a relation schema.
- **Relation key:** A relation **key** is an attribute or combination of attributes that uniquely identifies a tuple in a table.
- **Domain:** The domain of an attribute is the set of values that can be stored in that attribute. For example, if an attribute represents ages, its domain would be any age from 0 onward.

Types of Relational Models

There are generally three types of relational models in DBMS -

1. The One-to-Many Model

In this type of relationship, one entity is related to multiple other entities. For example, a customer may have multiple orders, a teacher may have multiple students, and a product may be sold in multiple stores. The key feature of the one-to-many model is that the relationship between two entities is not reciprocal. That is, the fact that a customer has multiple orders does not mean that an order has multiple customers.

2. The One-to-One Model

The one-to-one model of relational models states that there is a direct and unique relationship between two specific objects. In other words, each object can only be related to one other object and vice versa. This type of relationship is often used to represent relationships between physical things, such as people and their driver's license numbers. While the one-to-one model is relatively simple, it can be useful for representing complex real-world relationships.

3. The Many-to-Many Model

In a many-to-many model, each record in the primary table can be linked to multiple records in the foreign table and vice versa. This type of relationship is less common. For example, a many-to-many relationship takes place between students and courses. A student can take multiple courses, and a course can have multiple students enrolled. To create a many-to-many relationship, we need to create a third table, called a junction table, that contains the primary keys of the other tables. The junction table establishes the relationships between the rows in the two other tables.

Advantages of the Relational Model

- **Simple model:** Relational Model is simple and easy to use in comparison to other languages.
- **Flexible:** Relational Model is more flexible than any other relational model present.
- **Secure:** Relational Model is more secure than any other relational model.
- **Data Accuracy:** Data is more accurate in the relational data model.

- **Data Integrity:** The integrity of the data is maintained in the relational model.
- **Operations can be Applied Easily:** It is better to perform operations in the relational model.

Disadvantages of the Relational Model

- Relational Database Model is not very good for large databases.
- Sometimes, it becomes difficult to find the relation between tables.
- Because of the complex structure, the response time for queries is high.

Integrity Constraints

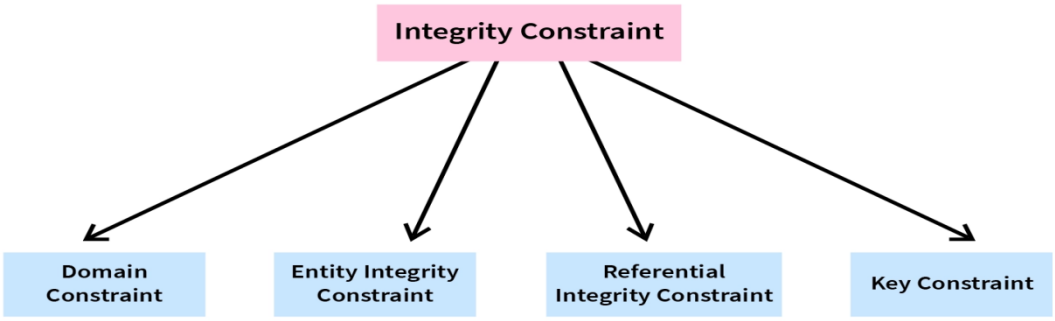
In Database Management Systems, integrity constraints are pre-defined set of rules that are applied on the table fields (columns) or relations to ensure that the overall validity, integrity, and consistency of the data present in the database table is maintained.

Evaluation of all the conditions or rules mentioned in the integrity constraint is done every time a table insert, update, delete, or alter operation is performed. The data can be inserted, updated, deleted, or altered only if the result of the constraint comes out to be true. Thus, integrity constraints are useful in preventing any accidental damage to the database by an authorized user.

Types of Integrity Constraints

There are four types of integrity constraints in DBMS:

1. Domain Constraint
2. Entity Constraint
3. Referential Integrity Constraint
4. Key Constraint



Domain Constraint

Domain integrity constraint contains a certain set of rules or conditions to restrict the kind of attributes or values a column can hold in the database table. The data type of a domain can be string, integer, character, DateTime, currency, etc.

Example:Consider a Student's table having Roll No, Name, Age, Class of students.

Roll No	Name	Age	Class
101	Adam	14	6
102	Steve	16	8
103	David	8	4
104	Bruce	18	12
105	Tim	6	A

In the above student's table, the value A in the last row last column violates the domain integrity constraint because the Class attribute contains only integer values while A is a character.

Entity Integrity Constraint

Entity Integrity Constraint is used to ensure that the primary key cannot be null. A primary key is used to identify individual records in a table and if the primary key has a null value, then we can't identify those records. There can be null values anywhere in the table except the primary key column.

Example:

Consider Employees table having Id, Name, and salary of employees

ID	Name	Salary
1101	Jackson	40000
1102	Harry	60000
1103	Steve	80000
1104	Ash	1800000
	James	36000

In the above employee's table, we can see that the ID column is the primary key and contains a null value in the last row which violates the entity integrity constraint.

Database System

Referential Integrity Constraint

Referential Integrity Constraint ensures that there must always exist a valid relationship between two relational database tables. This valid relationship between the two tables confirms that a foreign key exists in a table. It should always reference a corresponding value or attribute in the other table or be null.

Example:

Consider an Employee and a Department table where Dept_ID acts as a foreign key between the two tables:

Employee table

ID	Name	Salary	Dept_ID
1101	Jackson	40000	3
1102	Harry	60000	2
1103	Steve	80000	4
1104	Ash	1800000	3
1105	James	36000	1

Department table

Dept_ID	Dept_Name
1	Sales
2	HR
3	Technical

In the above example, Dept_ID acts as a foreign key in the Employees table and a primary key in the Department table. Row having *DeptID=4* violates the referential integrity constraint since DeptID 4 is not defined as a primary key column in the Departments table.

Key constraint

Keys are the set of entities that are used to identify an entity within its entity set uniquely. There could be multiple keys in a single entity set, but out of these multiple keys, only one key will be the primary key. A primary key can only contain unique and not null values in the relational database table.

Database System

Example:

Consider a student's table

Roll No	Name	Age	Class
101	Adam	14	6
102	Steve	16	8
103	David	8	4
104	Bruce	18	12
102	Tim	6	2

The last row of the student's table violates the key integrity constraint since Roll No 102 is repeated twice in the primary key column. A primary key must be unique and not null therefore duplicate values are not allowed in the Roll No column of the above student's table.

Database System

Difference between Database Management System and File Processing System

Basics	File Processing System	DBMS
Meaning	The users are not required to write procedures.	The user has to write procedures for managing databases.
Structure	The file system is a way of arranging the files in a storage medium within a computer.	DBMS is software for managing the database.
Data Abstraction	It give details of storage and representation of data	It hides the internal details of Database
Data Redundancy	Redundant data can be present in a file system.	In DBMS there is no redundant data.
Security Constraints	File systems provide less security in comparison to DBMS.	DBMS has more security mechanisms as compared to file systems.
Backup and Recovery	It doesn't provide Inbuilt mechanism for backup and recovery of data if it is lost.	It provides in house tools for backup and recovery of data even if it is lost.
Consistency	There is less data consistency in the file system.	There is more data consistency because of the process of normalization.
Complexity	It is less complex as compared to DBMS.	It has more complexity in handling as compared to the file system.
Cost	It is less expensive than DBMS.	It has a comparatively higher cost than a file system.
Example	Cobol, C++	Oracle, SQL Server

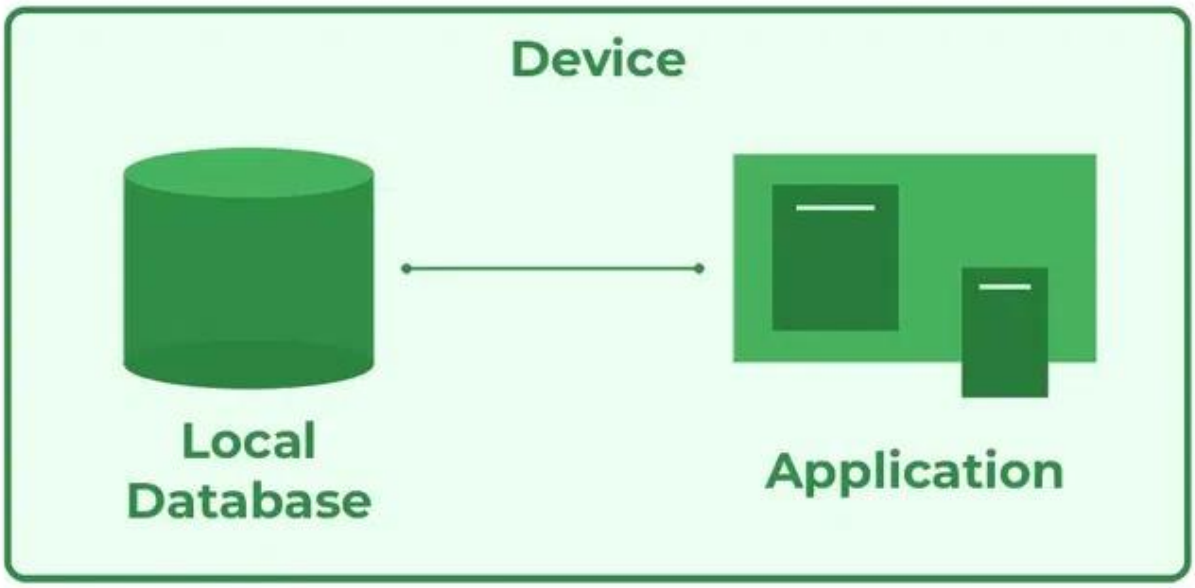
Database System

Database Architecture

The architecture of a database system (or DB architecture) describes how a database system interacts with your application. It tells you, for example, how data flows from a database to a web browser.

There are three database architecture types based on the number of tiers or levels between the client (software) and the database.

One Tier



This is one of the simplest database architecture types. Here, the client and database server are all located on one machine. So, for example, if you have a web page saved on your computer that retrieves data from a local database, that's a one-tier architecture. You can view one tier as a "zero tier" architecture. Database communications are often quick and direct since it's a local connection.

Database System

But, of course, that also means that the database can only serve one client – the user of that computer.

So, why would you want a one-tier architecture?

Well, this structure is only used during development and testing. It allows programmers to create database functions quickly without worrying about network lag. Network administrators and system operators also use this architecture if they want to work with the database directly.

The lack of security and the inability to service multiple clients make one-tier architectures unsuitable for production.

Two Tier

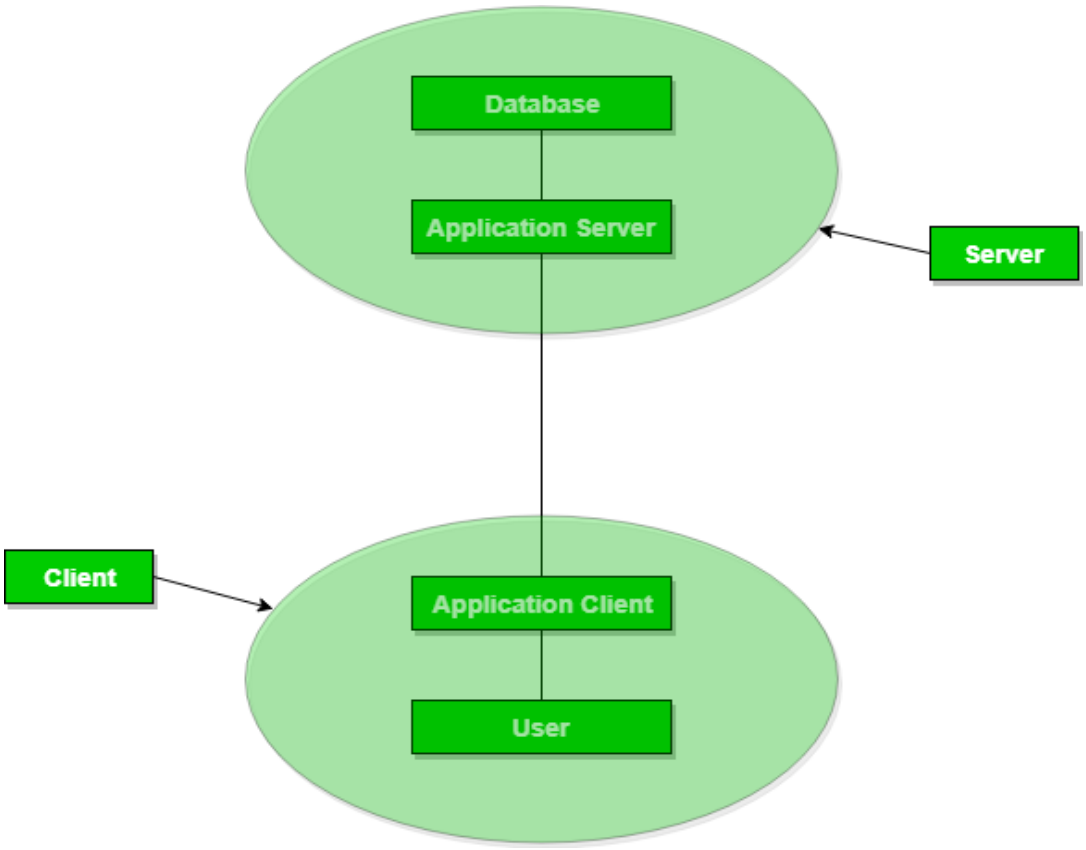
In two-tier architecture, the Database system is present at the server machine and the DBMS application is present at the client machine, these two machines are connected with each other through a reliable network as shown in the above diagram.

Whenever client machine makes a request to access the database present at server using a query language like sql, the server perform the request on the

Database System

database and returns the result back to the client. The application connection interface such as JDBC, ODBC are used for the interaction between server and client.

Three Tier



In three-tier architecture, another layer is present between the client machine and server machine. In this architecture, the client application doesn't communicate directly with the database systems present at the server machine, rather the client application communicates with server application and the server application internally communicates with the database system present at the server.

Database System

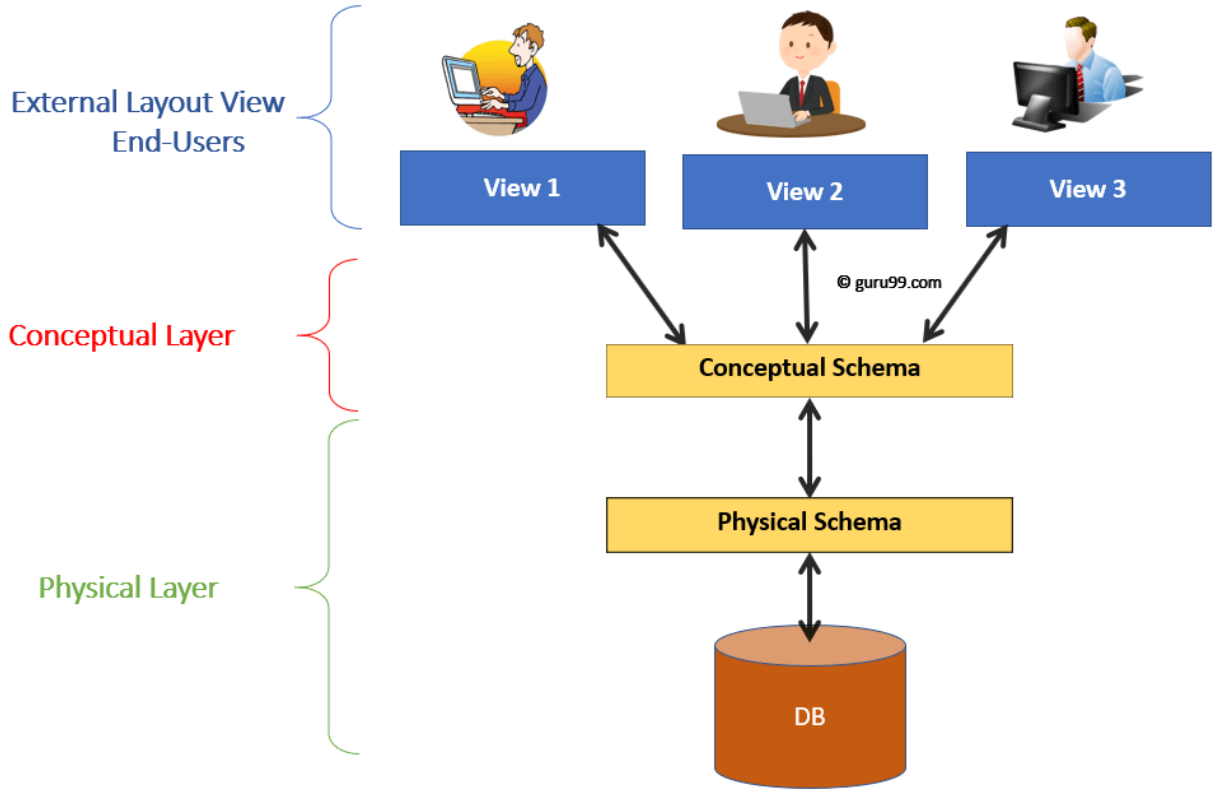
Three Schemas Architecture/Three level of Abstraction

This architecture has three levels:

1. External level

2. Conceptual level

3. Internal level



1. External level

It is also called **view level**. The reason this level is called “view” is because several users can view their desired data from this level which is internally fetched from database with the help of conceptual and internal level mapping. The user doesn’t need to know the database schema details such as data structure, table definition

etc. user is only concerned about data which is what returned back to the view level after it has been fetched from database (present at the internal level).

External level is the "**top level**" of the Three Level DBMS Architecture.

2. Conceptual level

It is also called **logical level**. The whole design of the database such as relationship among data, schema of data etc. are described in this level. Database constraints and security are also implemented in this level of architecture. This level is maintained by DBA (database administrator).

3. Internal level

This level is also known as physical level. This level describes how the data is actually stored in the storage devices. This level is also responsible for allocating space to the data. This is the lowest level of the architecture.

Entity Relationship Model

The Entity Relational Model is a model for identifying entities to be represented in the database and representation of how those entities are related. The ER data model specifies enterprise schema that represents the overall logical structure of a database graphically.

The Entity Relationship Diagram explains the relationship among the entities present in the database. ER models are used to model real-world objects like a person, a car, or a company and the relation between these real-world objects. In short, the ER Diagram is the structural format of the database.

Why Use ER Diagrams In DBMS?

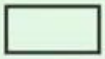
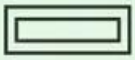
- ER diagrams are used to represent the E-R model in a database, which makes them easy to convert into relations (tables).
- ER diagrams provide the purpose of real-world modeling of objects which makes them intently useful.
- ER diagrams require no technical knowledge and no hardware support.
- These diagrams are very easy to understand and easy to create even for a naive user.
- It gives a standard solution for visualizing the data logically**.**

Symbols Used in ER Model

ER Model is used to model the logical view of the system from a data perspective which consists of these symbols:

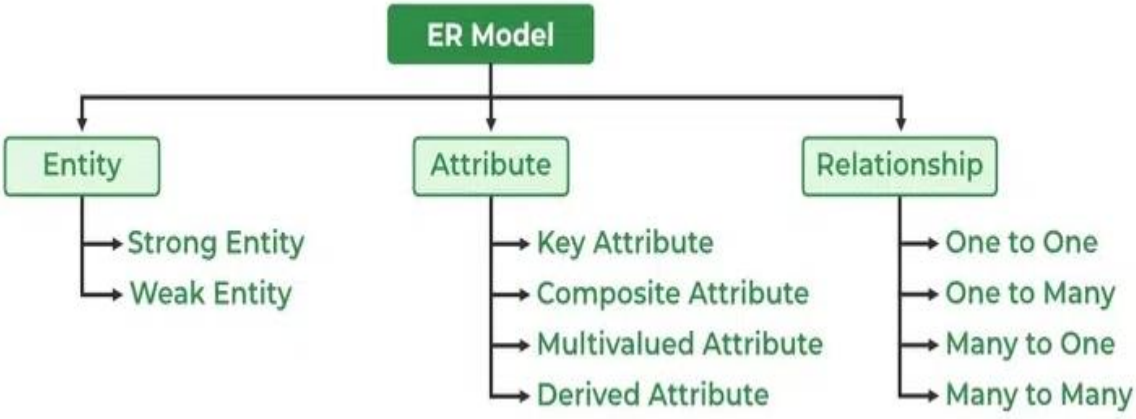
- **Rectangles:** Rectangles represent Entities in the ER Model.
- **Ellipses:** Ellipses represent Attributes in the ER Model.
- **Diamond**: Diamonds represent Relationships among Entities.

- **Lines:** Lines represent attributes to entities and entity sets with other relationship types.
- **Double Ellipse:** Double Ellipses represent Multi-Valued Attributes.
- **Double Rectangle:** Double Rectangle represents a Weak Entity.

Figures	Symbols	Represents
Rectangle		Entities in ER Model
Ellipse		Attributes in ER Model
Diamond		Relationships among Entities
Line		Attributes to Entities and Entity Sets with Other Relationship Types
Double Ellipse		Multi-Valued Attributes
Double Rectangle		Weak Entity

Components of ER Diagram

ER Model consists of Entities, Attributes, and Relationships among Entities in a Database System.

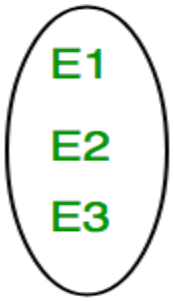


Entity

An Entity may be an object with a physical existence – a particular person, car, house, or employee – or it may be an object with a conceptual existence – a company, a job, or a university course.

Entity Set: An Entity is an object of Entity Type and a set of all entities is called an entity set. For Example, E1 is an entity having Entity Type Student and the set of all students is called Entity Set. In ER diagram, Entity Type is represented as:

Entity Type



1. Strong Entity

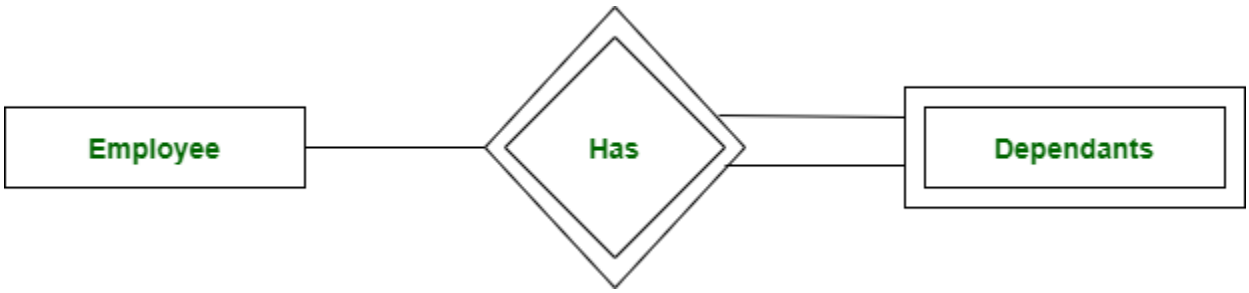
A Strong Entity is a type of entity that has a key Attribute. Strong Entity does not depend on other Entity in the Schema. It has a primary key, that helps in identifying it uniquely, and it is represented by a rectangle. These are called Strong Entity Types.

2. Weak Entity

An Entity type has a key attribute that uniquely identifies each entity in the entity set. But some entity type exists for which key attributes can't be defined. These are called Weak Entity types.

For Example, A company may store the information of dependents (Parents, Children, Spouse) of an Employee. But the dependents don't have existed without the employee. So Dependent will be a **Weak Entity Type** and Employee will be Identifying Entity type for Dependent, which means it is **Strong Entity Type**.

A weak entity type is represented by a Double Rectangle. The participation of weak entity types is always total. The relationship between the weak entity type and its identifying strong entity type is called identifying relationship and it is represented by a double diamond.



Attributes

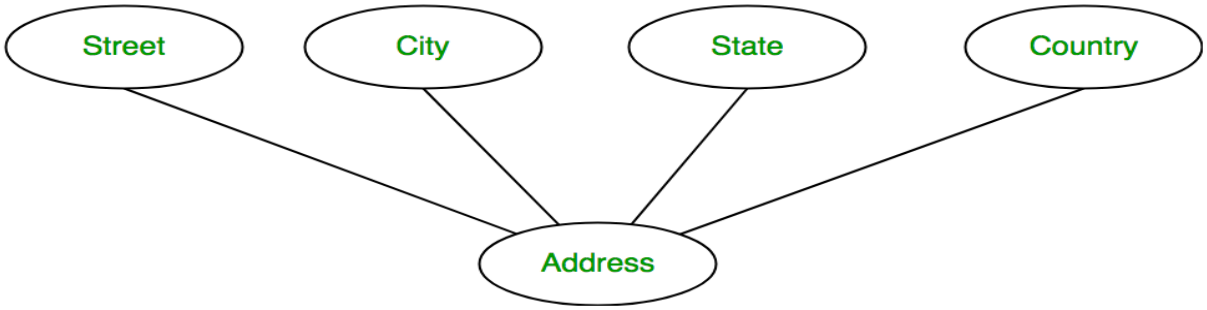
Attributes are the properties that define the entity type. For example, Roll_No, Name, DOB, Age, Address, and Mobile_No are the attributes that define entity type Student. In ER diagram, the attribute is represented by an oval.

1. Key Attribute

The attribute which **uniquely identifies each entity** in the entity set is called the key attribute. For example, Roll_No will be unique for each student. In ER diagram, the key attribute is represented by an oval with underlying lines.

2. Composite Attribute

An attribute **composed of many other attributes** is called a composite attribute. For example, the Address attribute of the student Entity type consists of Street, City, State, and Country. In ER diagram, the composite attribute is represented by an oval comprising of ovals.



3. Multivalued Attribute

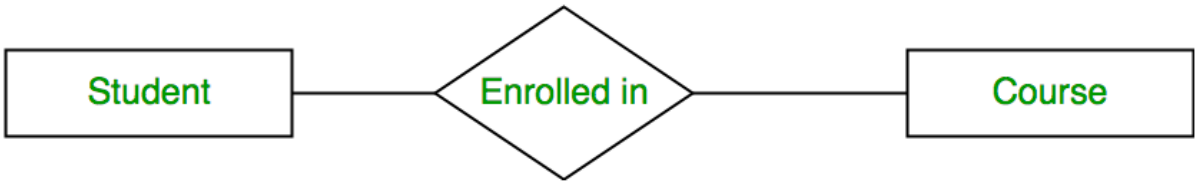
An attribute consisting of more than one value for a given entity. For example, Phone_No (can be more than one for a given student). In ER diagram, a multivalued attribute is represented by a double oval.

4. Derived Attribute

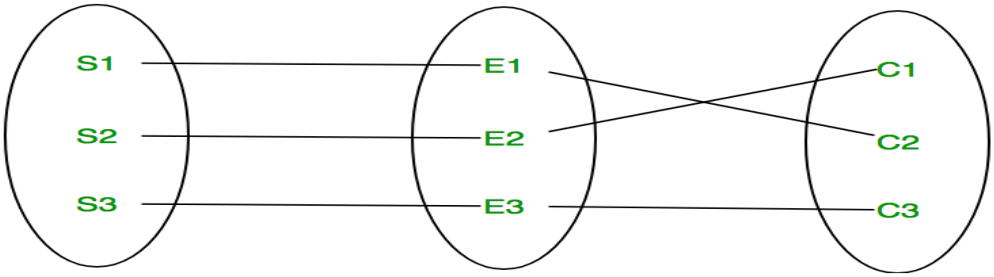
An attribute that can be derived from other attributes of the entity type is known as a derived attribute. e.g.; Age (can be derived from DOB). In ER diagram, the derived attribute is represented by a dashed oval.

Relationship Type and Relationship Set

A Relationship Type represents the association between entity types. For example, 'Enrolled in' is a relationship type that exists between entity type Student and Course. In ER diagram, the relationship type is represented by a diamond and connecting the entities with lines.



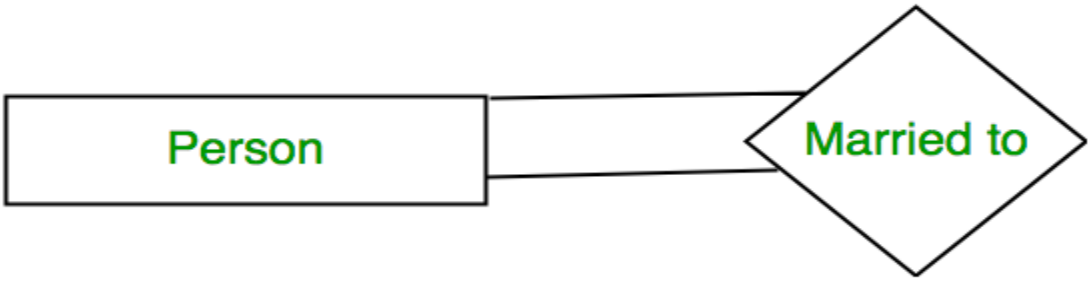
A set of relationships of the same type is known as a relationship set. The following relationship set depicts S1 as enrolled in C2, S2 as enrolled in C1, and S3 as registered in C3.



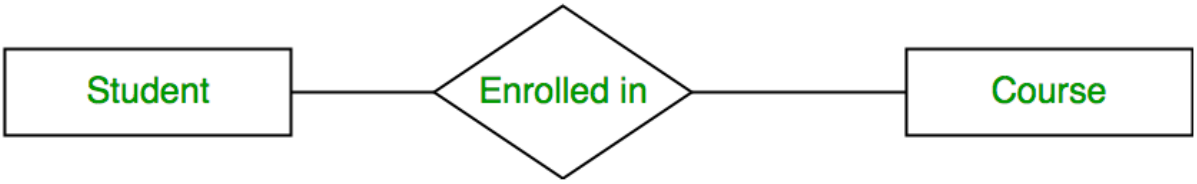
Degree of a Relationship Set

The number of different entity sets participating in a relationship set is called the degree of a relationship set.

1. **Unary Relationship:** When there is only ONE entity set participating in a relation, the relationship is called a unary relationship. For example, one person is married to only one person.



2. **Binary Relationship:** When there are TWO entities set participating in a relationship, the relationship is called a binary relationship. For example, a Student is enrolled in a Course.



Cardinality

The number of times an entity of an entity set participates in a relationship set is known as cardinality. Cardinality can be of different types:

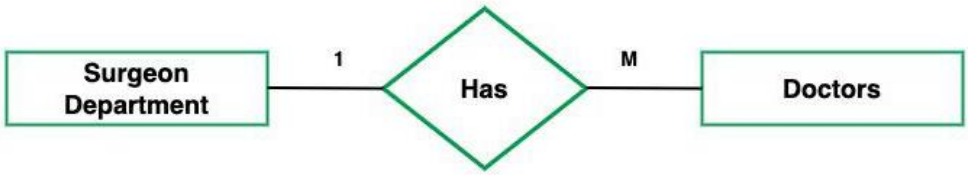
1. One-to-One: When each entity in each entity set can take part only once in the relationship, the cardinality is one-to-one. Let us assume that a male can marry one female and a female can marry one male. So the relationship will be one-to-one.

The total number of tables that can be used in this is 2.

2. One-to-Many: In one-to-many mapping as well where each entity can be related to more than one relationship and the total number of tables that can be used in this is 2. Let us assume that one surgeon department can

accommodate many doctors. So the Cardinality will be 1 to M. It means one department has many Doctors.

Total number of tables that can used is 3.

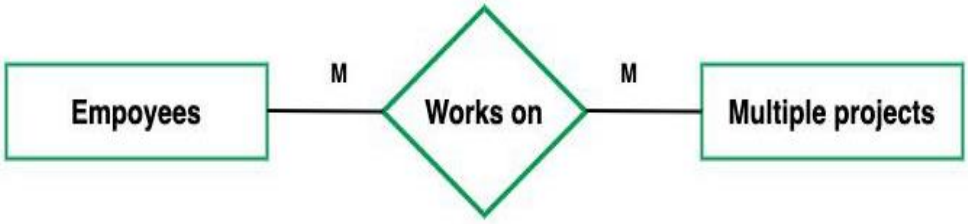


3. Many-to-One: When entities in one entity set can take part only once in the relationship set and entities in other entity sets can take part more than once in the relationship set, cardinality is many to one. Let us assume that a student can take only one course but one course can be taken by many students. So the cardinality will be n to 1. It means that for one course there can be n students but for one student, there will be only one course.

The total number of tables that can be used in this is 3.

Many-to-Many: When entities in all entity sets can take part more than once in the relationship cardinality is many to many. Let us assume that a student can take more than one course and one course can be taken by many students. So the relationship will be many to many.

The total number of tables that can be used in this is 3.



File based Approach:

This is the oldest system used for record keeping in organizations. In this system a set of books or files in prepared. Each book or file holds a particular set of information in the organization .Each department of the organization has a separate file for every specific task of department. The information is shared among different departments through files. Typically many such files are labeled and placed in one or more cabinets .For security the cabinets are locked or may be locked in secure areas of building .

Disadvantages of Manual File Based Approach

- **Slow Method**
- **Costly**
- **Complex**
- **Difficult in searching process**
- **Duplication of data**
- **In efficient**
- **Physical Damages**

File processing system

Before understanding the characteristics of a pure database system, it is necessary to know about the file processing system. A file processing system is a collection of files and programs that store ,access , modify and manage data in these files .A file processing system is also called a computer file processing system or traditional file processing system or computer file based system.

In early days of the computer there were no database .Data was stored in a computer files or disk. Data was stored and managed in data files through the file processing system. This system is still used in some small organization.

Disadvantages of File processing System:

1) Data Redundancy

Redundancy means multiple copies of the same data.

2)Data Inconsistency

Data Inconsistency means different files may contain different information about a particular object or person.

3) Data Isolation

Storages of data in various files.

4)Data Atomicity

Atomicity means that operations must complete or fail as a whole unit.

5)Program Data Dependency

Close relationship between data stored in files and the software programs that update and maintain those files. Any change in data organization or format requires a change in all the programs associated with those files.

6)Difficult Program Maintenance

7)Limited Data Sharing

8)Security Problems

Natural Joins :

Natural join is also considered a type of inner join but it does not use any comparison operator for join condition. It joins the table only when the two tables have at least one common attribute with same name and domain.

In the result of the Natural Join the common attribute only appears once.

It will be clearer with help of an example:

Natural Join: (R ⋈ S)

Where R is the first relation
S is the second relation

EMPLOYEE ⋈ DEPARTMENT

Let's say we want to join EMPLOYEE and DEPARTMENT relation with E_NO as a common attribute.

Notice, here E_NO has the same name in both the relations and also consists of the same domain, i.e., in both relations E_NO is a string.

EMPLOYEE

E_NO	E_NAME	CITY	EXPERIENCE
E-1	Ali	Faisalabad	04
E-2	Ahmad	Lahore	09
E-3	Umar	Karachi	03
E-4	Adnan	Islamabad	07

DEPARTMENT

D_NO	D_NAME	E_NO	MIN_EXPERIENCE
D-1	HR	E-1	03
D-2	IT	E-2	05
D-3	Marketing	E-3	02

Also, let's have the Cartesian product of the above two relations. It will be much easier to understand Join Operations.

E_NO	E_NAME	CITY	EXPERIENCE	D_NO	D_NAME	E_NO	MIN_EXPERIENCE
E-1	Ali	Faisalabad	04	D-1	HR	E-1	03
E-1	Ali	Faisalabad	04	D-2	IT	E-2	05
E-1	Ali	Faisalabad	04	D-3	Marketing	E-3	02
E-2	Ahmad	Lahore	09	D-1	HR	E-1	03
E-2	Ahmad	Lahore	09	D-2	IT	E-2	05
E-2	Ahmad	Lahore	09	D-3	Marketing	E-3	02
E-3	Umar	Karachi	03	D-1	HR	E-1	03
E-3	Umar	Karachi	03	D-2	IT	E-2	05
E-3	Umar	Karachi	03	D-3	Marketing	E-3	02
E-4	Adnan	Islamabad	07	D-1	HR	E-1	03
E-4	Adnan	Islamabad	07	D-2	IT	E-2	05
E-4	Adnan	Islamabad	07	D-3	Marketing	E-3	02

Solution :

E_NO	E_NAME	CITY	EXPERIENCE	D_NO	D_NAME	MIN_EXPERIENCE
E-1	Ali	Faisalabad	04	D-1	HR	03
E-2	Ahmad	Lahore	09	D-2	IT	05
E-3	Umar	Karachi	03	D-3	Marketing	02

But unlike the above operation, where we have two columns of E_NO, here we are having only one column of E_NO. This is because Natural Join automatically keeps a single copy of a common attribute.

Database Systems

Database Systems

Normalization

- Normalization is the process of organizing the data in the database.
- Normalization is used to minimize the redundancy from a relation or set of relations. It is also used to eliminate undesirable characteristics like Insertion, Update, and Deletion Anomalies.
- Normalization divides the larger table into smaller and links them using relationships.
- The normal form is used to reduce redundancy from the database table.

Why do we need Normalization?

The main reason for normalizing the relations is removing these anomalies. Failure to eliminate anomalies leads to data redundancy and can cause data integrity and other problems as the database grows. Normalization consists of a series of guidelines that helps to guide you in creating a good database structure.

How Are Anomalies Caused in DBMS?

Database anomalies are the faults in the database caused due to poor management of storing everything in the flat database. It can be removed with the process of Normalization, which generally splits the database which results in reducing the anomalies in the database.

Data modification anomalies can be categorized into three types:

- **Insertion Anomaly:** Insertion Anomaly refers to when one cannot insert a new tuple into a relationship due to lack of data.
- **Deletion Anomaly:** The delete anomaly refers to the situation where the deletion of data results in the unintended loss of some other important data.

- **Updatation Anomaly:** The update anomaly is when an update of a single data value requires multiple rows of data to be updated.

Employee Table

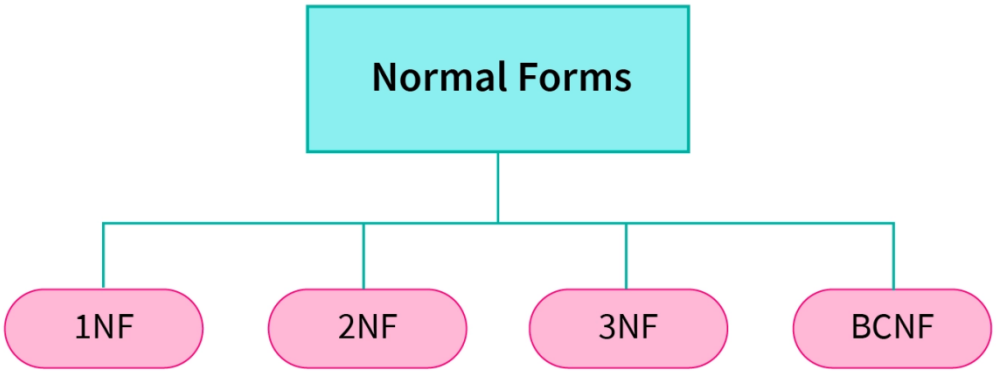
EMP_id	EMP_Name	EMP_dept	EMP_City
1	Hira	D001	Faisalabad
2	Umar	D001	Faisalabad
3	Ali	D001	Karachi
4	Saba	D001	Islamabad
5	Ahmad	D004	Lahore
6	Arham	D005	Lahore
7	Subhan	D004	Karachi

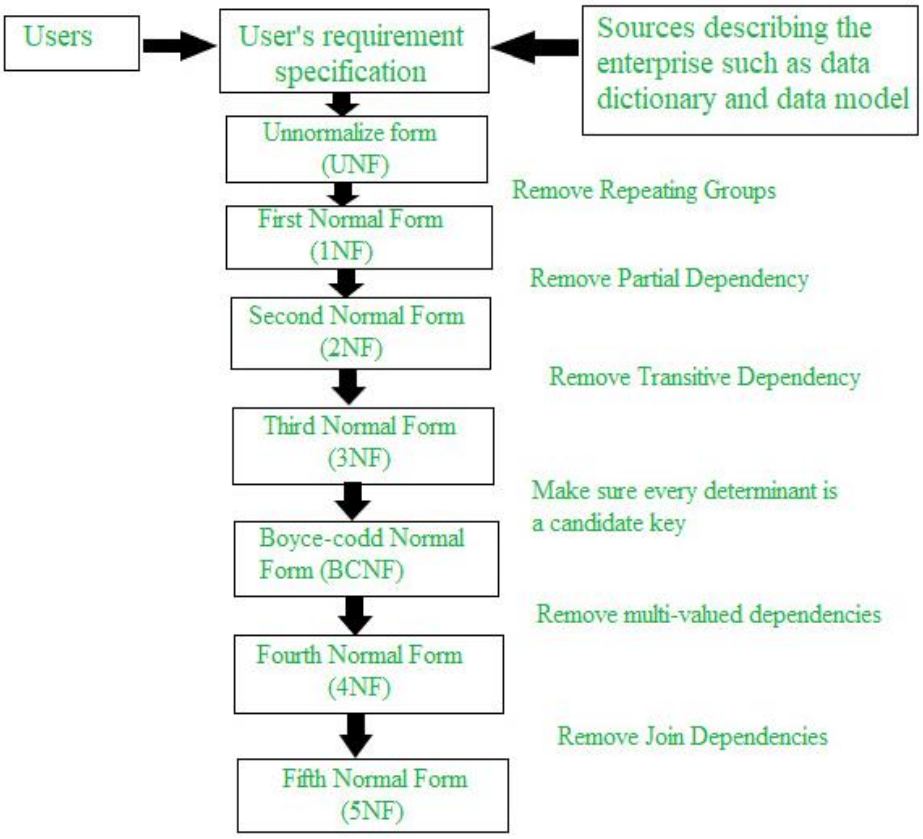
Types of Normal Forms:

Normalization works through a series of stages called Normal forms. The normal forms apply to individual relations. The relation is said to be in particular normal form if it satisfies constraints.

Following are the various types of Normal forms: **Normal Forms**

There are four types of normal forms that are usually used in relational databases as you can see in the following figure:


Normal Forms
1NF
2NF
3NF
BCNF


Users
User's requirement specification
Sources describing the enterprise such as data dictionary and data model
Unnormalize form (UNF)
Remove Repeating Groups
First Normal Form (1NF)
Remove Partial Dependency
Second Normal Form (2NF)
Remove Transitive Dependency
Third Normal Form (3NF)
Make sure every determinant is a candidate key
Boyce-codd Normal Form (BCNF)
Remove multi-valued dependencies
Fourth Normal Form (4NF)
Remove Join Dependencies
Fifth Normal Form (5NF)

Normal Form	Description
1NF	A relation is in 1NF if it contains an atomic value.
2NF	A relation will be in 2NF if it is in 1NF and all non-key attributes are fully functional dependent on the primary key.
3NF	A relation will be in 3NF if it is in 2NF and no transition dependency exists.
BCNF	A stronger definition of 3NF is known as Boyce Codd's normal form.
4NF	A relation will be in 4NF if it is in Boyce Codd's normal form and has no multi-valued dependency.
5NF	A relation is in 5NF. If it is in 4NF and does not contain any join dependency, joining should be lossless.

Functional Dependency

The functional dependency is a relationship that exists between two attributes. It typically exists between the primary key and non-key attribute within a table.

$X \rightarrow Y$

The left side of FD is known as a determinant, the right side of the production is known as a dependent.

For example:

Assume we have an employee table with attributes: Emp_Id, Emp_Name, Emp_Address.

Here Emp_Id attribute can uniquely identify the Emp_Name attribute of employee table because if we know the Emp_Id, we can tell that employee name associated with it.

Database Systems

Functional dependency can be written as:

Emp_Id → Emp_Name

Types of Functional Dependencies in DBMS

1. Trivial functional dependency
2. Non-Trivial functional dependency
3. Multivalued functional dependency
4. Transitive functional dependency

Trivial Functional dependency

In Trivial functional dependency, a dependent is always a subset of the determinant. In other words, a functional dependency is called trivial if the attributes on the right side are the subset of the attributes on the left side of the functional dependency.

X → Y is called a trivial functional dependency if Y is the subset of X.

For example, consider the Employee table below.

Employee_Id	Name	Age
1	Zayn	24
2	Ahmad	34
3	Ali	26
4	Umar	29

Here, **{ Employee_Id, Name } → { Name }** is a Trivial functional dependency, since the dependent **Name** is the subset of determinant **{ Employee_Id, Name }**.

{ Employee_Id } → { Employee_Id }, **{ Name } → { Name }** and **{ Age } → { Age }** are also Trivial.

Non-Trivial Functional Dependency in DBMS

- It is the opposite of Trivial functional dependency. Formally speaking, in **Non-Trivial functional dependency**, dependent if **not a subset** of the determinant.
- **X → Y** is called a Non-trivial functional dependency if **Y** is **not a subset** of **X**. So, a functional dependency **X → Y** where **X** is a set of attributes and **Y** is also a set of the attribute but not a subset of **X**, then it is called Non-trivial functional dependency.

For example, consider the Employee table below.

Employee_Id	Name	Age
1	Ahmad	24
2	Ali	34
3	Umar	26
4	Umair	29

- Here, **{ Employee_Id } → { Name }** is a non-trivial functional dependency because **Name**(dependent) is **not a subset** of **Employee_Id**(determinant).
- Similarly, **{ Employee_Id, Name } → { Age }** is also a non-trivial functional dependency.

Multivalued Functional Dependency in DBMS

In Multivalued functional dependency, attributes in the dependent set are not dependent on each other.

For example, X → {Y, Z}, if there exists is no functional dependency between Y and Z, then it is called as Multivalued functional dependency.

For example, consider the Employee table below.

Employee_Id	Name	Age
1	Umar	24
2	Umair	34
3	Ahmad	26
4	Ali	29
4	Imran	24

Here, **{ Employee_Id } → { Name, Age }** is a Multivalued functional dependency, since the dependent attributes **Name, Age** are not *functionally dependent*(**i.e. Name → Age or Age → Name doesn't exist !**).

Transitive Functional Dependency in DBMS

- Consider two functional dependencies **A → B** and **B → C** then according to the *transitivity axiom* **A → C** must also exist. This is called a transitive functional dependency.
- In other words, *dependent is indirectly dependent on determinant in Transitive functional dependency.*
- For example, consider the *Employee* table below.
- Here, **{ Employee_Id → Department }** and **{ Department → Street Number }** holds true. *Hence*, according to the **axiom of transitivity**, **{ Employee_Id → Street Number }** is a valid functional dependency.

Employee_Id	Name	Department	Street Number
1	Zayn	CD	11
2	Umar	AB	24
3	Umair	CD	11
4	Ali	PQ	71
5	Ahmad	LM	21

- Here, **{ Employee_Id → Department }** and **{ Department → Street Number }** holds true. *Hence*, according to the **axiom of transitivity**, **{ Employee_Id → Street Number }** is a valid functional dependency.

Properties of Functional Dependency in DBMS

William Armstrong in *1974* suggested a few rules related to functional dependency. They are called RAT rules.

1. **Reflexivity:** If A is a set of attributes and B is a subset of A, then the functional dependency A → B holds true.
    - *For example*, { Employee_Id, Name } → Name is valid.
2. **Augmentation:** If a functional dependency A → B holds true, then appending any number of the attribute to both sides of dependency doesn't affect the dependency. It remains true.
    - *For example,* X → Y holds true then, ZX → ZY also holds true.
    - *For example,* if { Employee_Id, Name } → { Name } holds true then, { Employee_Id, Name, Age } → { Name, Age }
3. **Transitivity:** If two functional dependencies X → Y and Y → Z hold true, then X → Z also holds true by the rule of Transitivity.

- *For example,* if { Employee_Id } → { Name } holds true and { Name } → { Department } holds true, then { Employee_Id } → { Department } also holds true.

Advantages of Functional Dependency in DBMS

Some of the advantages of Functional dependency,

1. It is used to maintain the quality of data in the database.
2. It expresses the facts about the database design.
3. It helps in clearly defining the meanings and constraints of databases.
4. It helps to identify bad designs.
5. Functional Dependency removes data redundancy where the same values should not be repeated at multiple locations in the same database table.
6. The process of Normalization starts with identifying the candidate keys in the relation. Without functional dependency, it's impossible to find candidate keys and normalize the database.

Normal Forms

First Normal Form (1NF)

A relation is in 1NF if every attribute is a single-valued attribute or it does not contain any multi-valued **or composite attribute**, i.e., every attribute is an atomic attribute.A relation is in first normal form if it does not contain a repeating groups. If there is a composite or multi-valued attribute, it violates the 1NF. To solve this, user can create a new row for each of the values of the multi-valued attribute to convert the table into the 1NF.

Let’s take an example of a relational table <EmployeeDetail> that contains the details of the employees of the company.

<EmployeeDetail>

Employee Code	Employee Name	Employee Phone Number
101	Ahmad	98765623,998234123
101	Ali	89023467
102	Umar	76213908
103	Imran	98132452

Here, the Employee Phone Number is a multi-valued attribute. So, this relation is not in 1NF.

To convert this table into 1NF, Make new rows with each Employee Phone Number as a new row as shown below:

<EmployeeDetail>

Employee Code	Employee Name	Employee Phone Number
101	Ahmad	998234123
101	Ahmad	98765623
101	Ali	89023467
102	Umar	76213908
103	Imran	98132452

Second Normal Form (2NF)

The normalization of 1NF relations to 2NF involves the elimination of partial dependencies. A Partial dependency in DBMS exists when any non-prime attributes, i.e., an attribute not a part of the candidate key, is not fully functionally dependent on one of the candidate keys.

For a relational table to be in second normal form, it must satisfy the following rules:

1. The table must be in first normal form.
2. It must not contain any partial dependency, i.e., all non-prime attributes are fully functionally dependent on the primary key.

If a partial dependency exists, we can divide the table to remove the partially dependent attributes and move them to some other table where they fit in well.

Let us take an example of the following <EmployeeProjectDetail> table to understand what partial dependency is and how to normalize the table to the second normal form:

<EmployeeProjectDetail>

Employee Code	Project ID	Employee Name	Project Name
101	P03	Ahmad	Project103
101	P01	Ali	Project101
102	P04	Rehan	Project104
103	P02	Imran	Project102

In the above table, the prime attributes of the table are Employee Code and Project ID. We have partial dependencies in this table because Employee Name can be determined by Employee Code and Project Name can be determined by Project ID. Thus, the above relational table violates the rule of 2NF.

The prime attributes in DBMS are those which are part of one or more candidate keys.

To remove partial dependencies from this table and normalize it into second normal form, Decompose the <EmployeeProjectDetail> table into the following three tables:

<EmployeeDetail>

Employee Code	Employee Name
101	Ahmad
101	Ali
102	Rehan
103	Imran

<EmployeeProject>

Employee Code	Project ID
101	P03
101	P01
102	P04
103	P02

<ProjectDetail>

Project ID	Project Name
P03	Project103
P01	Project101
P04	Project104
P02	Project102

Thus, converted the <EmployeeProjectDetail> table into 2NF by decomposing it into <EmployeeDetail>, <ProjectDetail> and <EmployeeProject> tables. The above tables satisfy the following two rules of 2NF as they are in 1NF and every non-prime attribute is fully dependent on the primary key.

The relations in 2NF are clearly less redundant than relations in 1NF. However, the decomposed relations may still suffer from one or more anomalies due to the transitive dependency. The transitive dependencies will remove in the Third Normal Form.

Third Normal Form (3NF)

The normalization of 2NF relations to 3NF involves the elimination of transitive dependencies in DBMS.

A functional dependency X -> Z is said to be transitive if the following three functional dependencies hold:

- X -> Y
- Y does not -> X
- Y -> Z

For a relational table to be in third normal form, it must satisfy the following rules:

1. The table must be in the second normal form.
2. No non-prime attribute is transitively dependent on the primary key.
3. For each functional dependency X -> Z at least one of the following conditions hold:

- X is a super key of the table.
- Z is a prime attribute of the table.

If a transitive dependency exists, we can divide the table to remove the transitively dependent attributes and place them to a new table along with a copy of the determinant.

Let us take an example of the following <EmployeeDetail> table to understand what is transitive dependency and how to normalize the table to the third normal form:

<EmployeeDetail>

Employee Code	Employee Name	Employee Zipcode	Employee City
101	Ali	110033	Faisalabad
101	Ahmad	110044	Lahore
102	Rehan	110028	Karachi
103	Imran	110064	Islamabad

The above table is not in 3NF because it has Employee Code -> Employee City transitive dependency because:

- Employee Code -> Employee Zipcode
- Employee Zipcode -> Employee City

Also, Employee Zipcode is not a super key and Employee City is not a prime attribute.

To remove transitive dependency from this table and normalize it into the third normal form, we can decompose the <EmployeeDetail> table into the following two tables:

<EmployeeDetail>

Employee Code	Employee Name	Employee Zipcode
101	Ahmad	110033
101	Ahmad	110044
102	Ali	110028
103	Rehan	110064

<EmployeeLocation>

Employee Zipcode	Employee City
110033	Model Town
110044	Badarpur
110028	Naraina
110064	Hari Nagar

Thus, converted the <EmployeeDetail> table into 3NF by decomposing it into <EmployeeDetail> and <EmployeeLocation> tables as they are in 2NF and they don't have any transitive dependency.

The 2NF and 3NF impose some extra conditions on dependencies on candidate keys and remove redundancy caused by that. However, there may still exist some dependencies that cause redundancy in the database. These redundancies are removed by a more strict normal form known as BCNF.

Boyce-Codd Normal Form (BCNF)

Boyce-Codd Normal Form(BCNF) is an advanced version of 3NF as it contains additional constraints compared to 3NF.

For a relational table to be in Boyce-Codd normal form, it must satisfy the following rules:

1. The table must be in the third normal form.
2. For every non-trivial functional dependency X -> Y, X is the superkey of the table. That means X cannot be a non-prime attribute if Y is a prime attribute.

A super key is a set of one or more attributes that can uniquely identify a row in a database table.

Let us take an example of the following <EmployeeProjectLead> table to understand how to normalize the table to the BCNF:

<EmployeeProjectLead>

Employee Code	Project ID	Project Leader
101	P03	Umar
101	P01	Ali
102	P04	Imran
103	P02	Rehan

The above table satisfies all the normal forms till 3NF, but it violates the rules of BCNF because the candidate key of the above table is {Employee Code, Project ID}. For the non-trivial functional dependency, Project Leader -> Project ID, Project ID is a prime attribute but Project Leader is a non-prime attribute. This is not allowed in BCNF.

To convert the given table into BCNF, Dcompose it into three tables:

<EmployeeProject>

Employee Code	Project ID
101	P03
101	P01
102	P04
103	P02

<ProjectLead>

Project Leader	Project ID
Umar	P03
Ali	P01
Imran	P04
Rehan	P02

Thus, converted the <EmployeeProjectLead> table into BCNF by decomposing it into <EmployeeProject> and <ProjectLead> tables.

Database System

Relation Keys

Keys are very important part of Relational database model. They are used to establish and identify relationships between tables and also to uniquely identify any record or row of data inside a table.

A Key can be a single attribute or a group of attributes, where the combination may act as a key.

Why we need a Key?

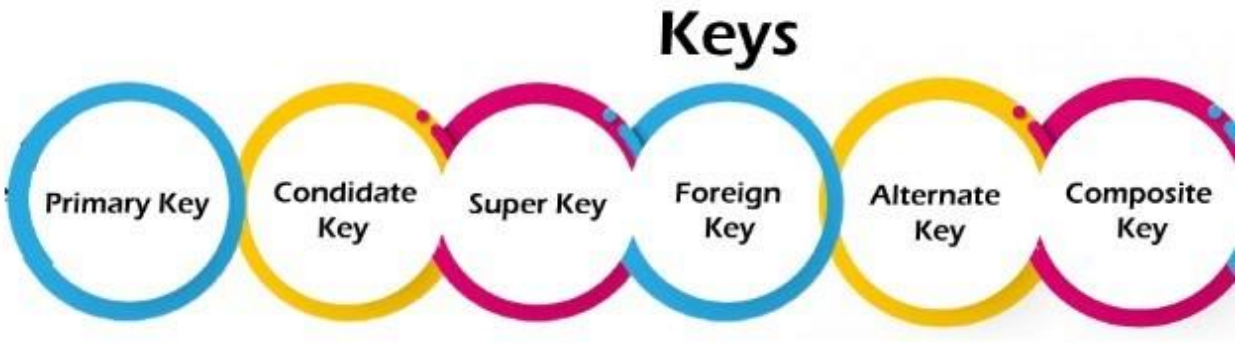
In real world applications, number of tables required for storing the data is huge, and the different tables are related to each other as well.

Also, tables store a lot of data in them. Tables generally extend to thousands of records stored in them, unsorted and unorganized.

Now to fetch any particular record from such dataset, user will have to apply some conditions, but what if there is duplicate data present and every time user try to fetch some data by applying certain condition, user get the wrong data. How many trials before get the right data?

To avoid all this, **Keys** are defined to easily identify any row of data in a table.

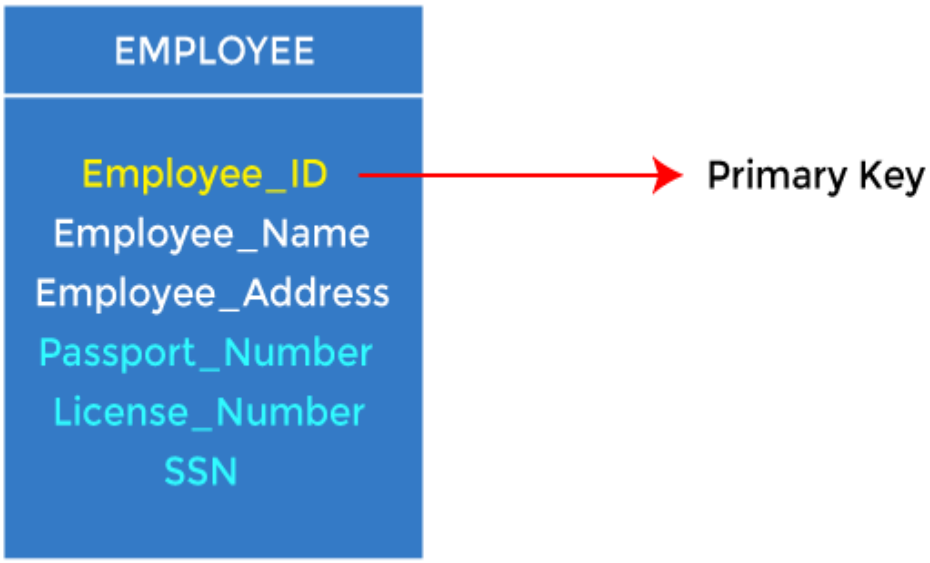
Types of keys



Database System

1. Primary key

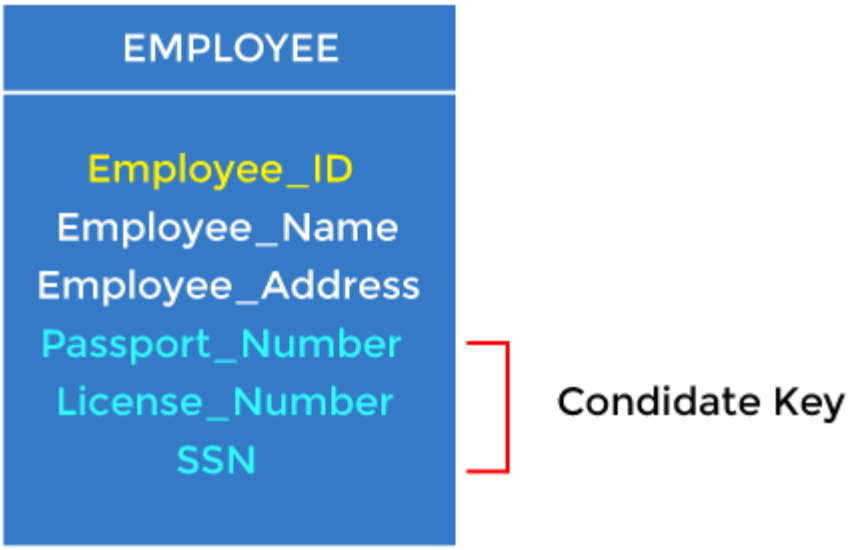
- It is the first key used to identify one and only one instance of an entity uniquely. An entity can contain multiple keys, as we saw in the PERSON table. The key which is most suitable from those lists becomes a primary key.
- In the EMPLOYEE table, ID can be the primary key since it is unique for each employee. In the EMPLOYEE table, we can even select License_Number and Passport_Number as primary keys since they are also unique.
- For each entity, the primary key selection is based on requirements and developers.



2. Candidate key

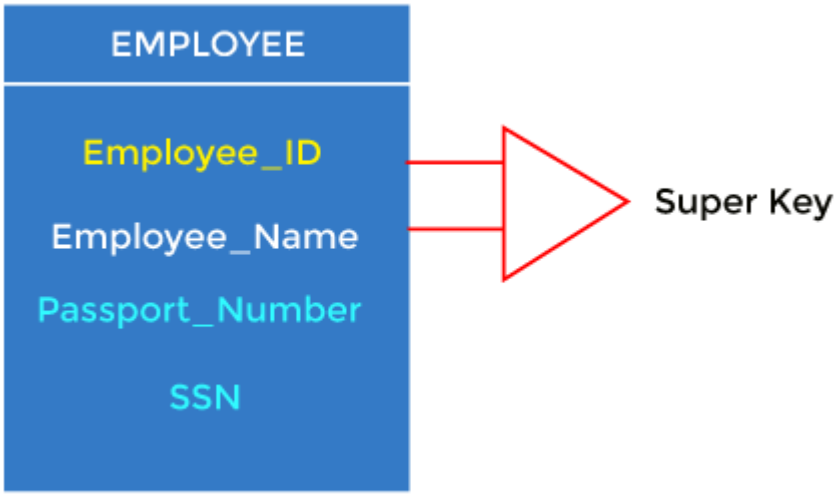
- A candidate key is an attribute or set of attributes that can uniquely identify a tuple.
- Except for the primary key, the remaining attributes are considered a candidate key. The candidate keys are as strong as the primary key.

For example: In the EMPLOYEE table, id is best suited for the primary key. The rest of the attributes, like SSN, Passport_Number, License_Number, etc., are considered a candidate key.



3. Super Key

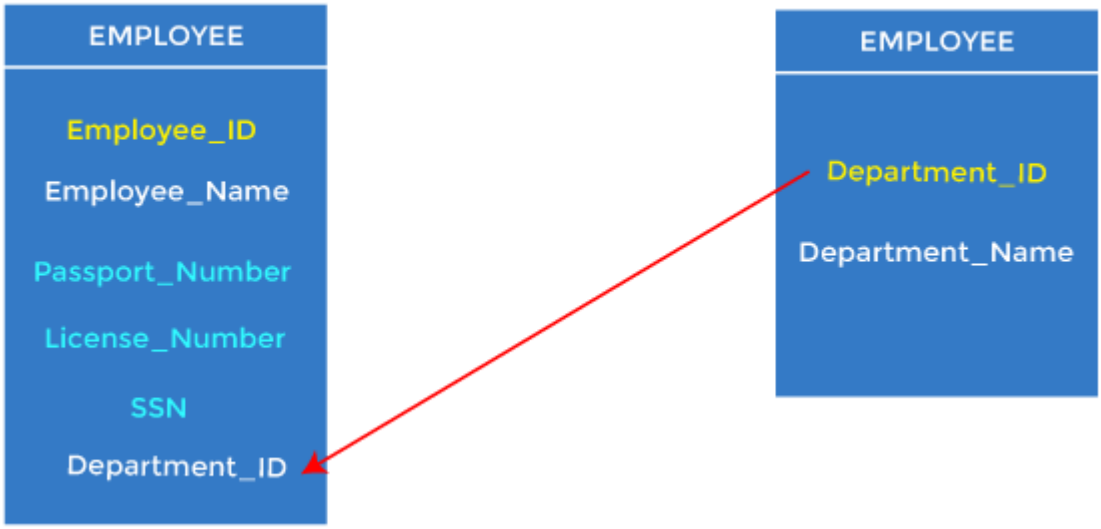
Super key is an attribute set that can uniquely identify a tuple. A super key is a superset of a candidate key.



For example: In the above EMPLOYEE table, for(EMPLOEE_ID, EMPLOYEE_NAME), the name of two employees can be the same, but their EMPLYEE_ID can't be the same. Hence, this combination can also be a key. The super key would be EMPLOYEE-ID (EMPLOYEE_ID, EMPLOYEE-NAME), etc.

4. Foreign key

- Foreign keys are the column of the table used to point to the primary key of another table.
- Every employee works in a specific department in a company, and employee and department are two different entities. So we can't store the department's information in the employee table. That's why we link these two tables through the primary key of one table.
- We add the primary key of the DEPARTMENT table, Department_Id, as a new attribute in the EMPLOYEE table.
- In the EMPLOYEE table, Department_Id is the foreign key, and both the tables are related.

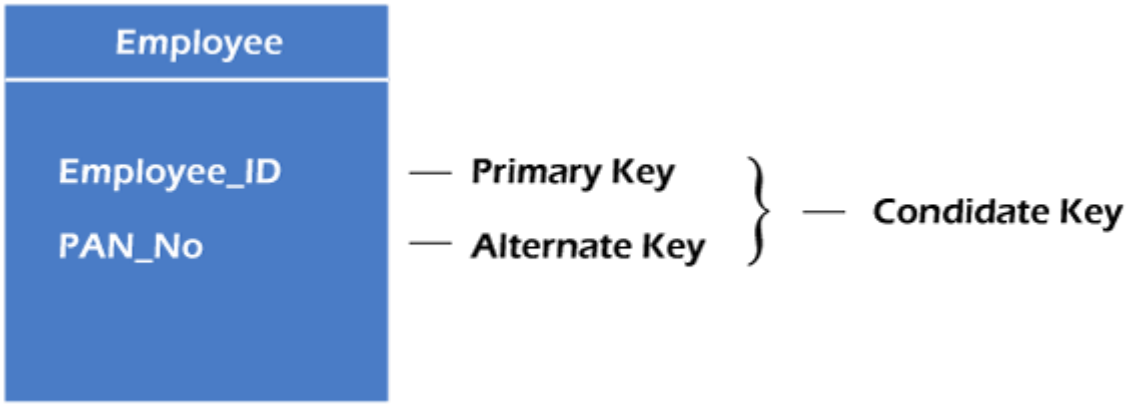


5. Alternate key

There may be one or more attributes or a combination of attributes that uniquely identify each tuple in a relation. These attributes or combinations of the attributes are called the candidate keys. One key is chosen as the primary key from these candidate keys, and the remaining candidate key, if it exists, is termed the alternate key. **In other words,** the total number of the alternate keys is the total number of candidate keys minus the primary key. The alternate key may or

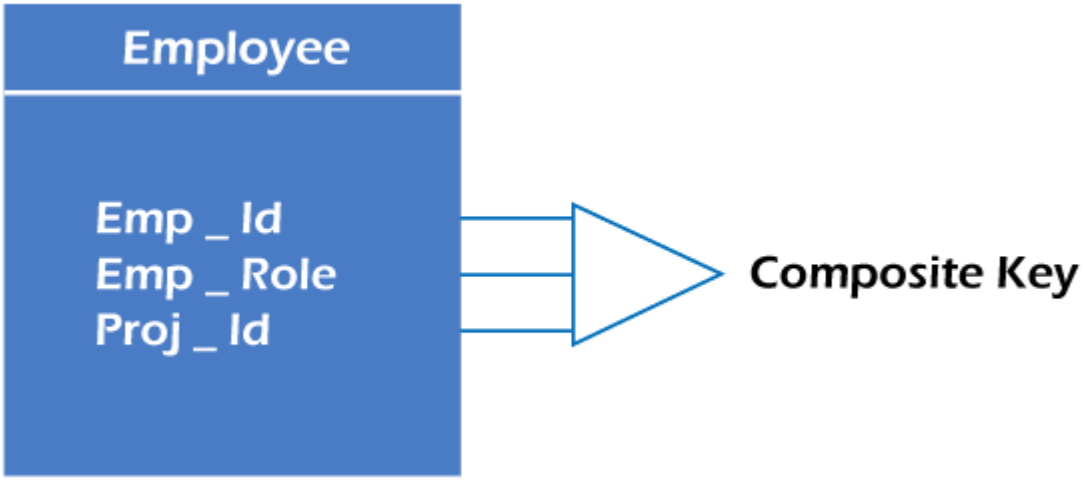
may not exist. If there is only one candidate key in a relation, it does not have an alternate key.

For example, employee relation has two attributes, Employee_Id and PAN_No, that act as candidate keys. In this relation, Employee_Id is chosen as the primary key, so the other candidate key, PAN_No, acts as the Alternate key.



6. Composite key

Whenever a primary key consists of more than one attribute, it is known as a composite key. This key is also known as Concatenated Key.



For example, in employee relations, we assume that an employee may be assigned multiple roles, and an employee may work on multiple projects

simultaneously. So the primary key will be composed of all three attributes, namely Emp_ID, Emp_role, and Proj_ID in combination. So these attributes act as a composite key since the primary key comprises more than one attribute.

Database Schema

- A database schema is the logical representation of a database, which shows how the data is stored logically in the entire database. It contains list of attributes and instruction that informs the database engine that how the data is organized and how the elements are related to each other.
- A database schema contains schema objects that may include **tables, fields, packages, views, relationships, primary key, foreign key,**
- In actual, the data is physically stored in files that may be in unstructured form, but to retrieve it and use it, user need to put it in a structured form. To do this, a database schema is used. It provides knowledge about how the data is organized in a database and how it is associated with other data. For example:

Student table:

Roll #	Name	Address

Course:

Course id	Name	Duration

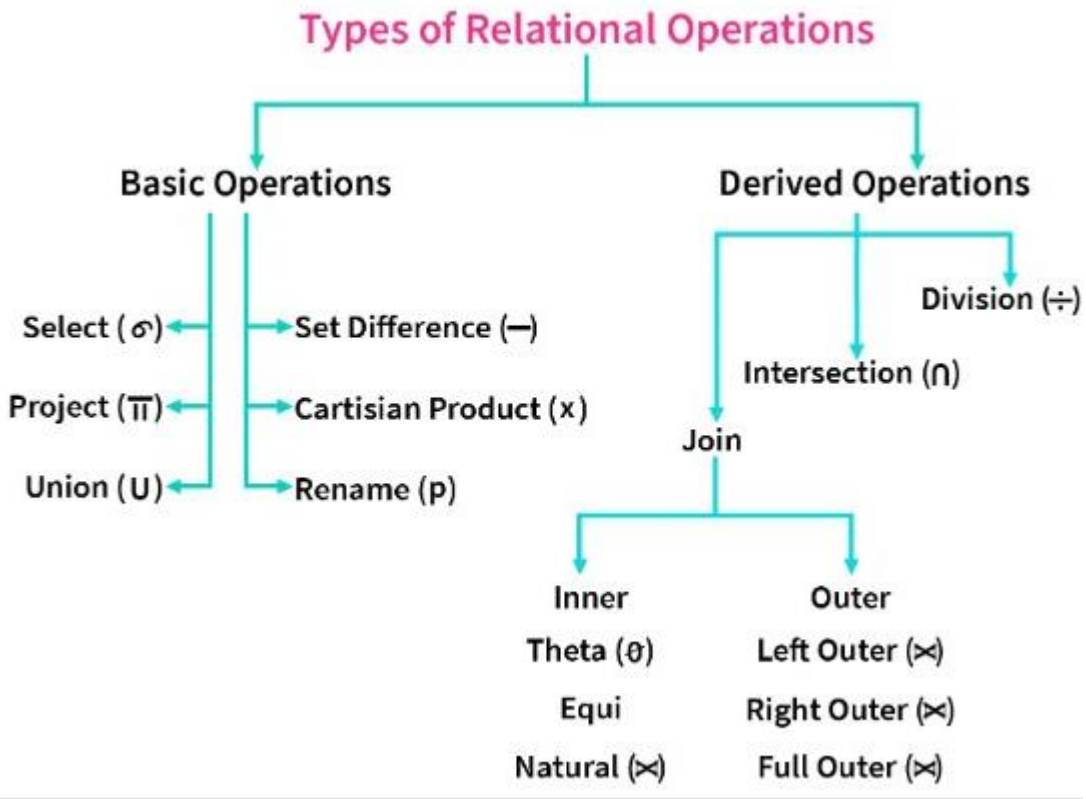
Relational Algebra

Relational algebra in DBMS is a procedural query language. Queries in relational algebra are performed using operators. Relational Algebra is the fundamental block for modern language SQL and modern Database Management Systems such as Oracle Database, Mircosoft SQL Server, IBM Db2, etc.

Relational Algebra came in 1970 and was given by Edgar F. Codd (Father of DBMS). It is also known as Procedural Query Language(PQL) as in PQL, a programmer/user has to mention two things, **"What to Do"** and **"How to Do"**.

Suppose our data is stored in a database, then relational algebra is used to access the data from the database.

Types of Relational Operations in DBMS



Database Systems

Select (σ)

This operation is used to select a subset of rows from a table based on a specific condition. Select operation is done by Selection Operator which is represented by "sigma"(σ). It is used to retrieve tuples (rows) from the table where the given condition is satisfied. It is a **unary operator** means it requires only one operand.
Notation:σP(R)

- Where σ is used to represent SELECTION
- R is used to represent RELATION
- p is the logic formula

Let's understand this with an example:

Roll No	Name	Age
1	Arman	20
2	Rehan	23
3	Hamza	26
4	Ahmad	20
5	Ali	24
6	Ahmad	22

Student table

Employee ID	Name	Age
E1	Ali	20
E2	Hamza	23
E3	Adnan	26
E4	Umar	20
E5	Ahmad	24

Employee Table

Suppose we want the row(s) from employee Relation where "AGE" is 20

Query: σ AGE=20 (Employee)

Solution:

Output:

Employee ID	Name	Age
E1	Ali	20
E4	Umar	20

Project (∏)

Project operation is done by Projection Operator which is represented by "pi"(∏). It is used to retrieve certain attributes(columns) from the table. It is also known as vertical partitioning as it separates the table vertically. It is also a unary operator.
Notation: ∏ a(r)

- Where ∏ is used to represent PROJECTION
- r is used to represent RELATION
- a is the attribute list

Let's understand this with an example:
Suppose we want the names of all students from STUDENT Relation.

Query :∏ NAME(STUDENT)

Output:

Name
Arman
Rehan
Hamza
Ahmad
Ali

From the above output it **eliminates duplicates**.
For multiple attributes, we can separate them using a ",".

∏ ROLL, NAME(STUDENT)

Output:

Roll No	Name
1	Arman
2	Rehan
3	Hamza
4	Ahmad
5	Ali
6	Ahmad

Union (∪)

Union operation is done by Union Operator which is represented by "union"(∪). It is the same as the union operator from set theory, i.e., it selects all tuples from both relations but with the exception that for the union of two relations/tables both relations must have the same set of Attributes. It is a binary operator as it requires two operands.

Notation: R ∪ S

- Where R is the first relation
- S is the second relation

If relations don't have the same set of attributes, then the union of such relations will result in NULL. Let's have an example to clarify the concept:
Suppose we want all the names from STUDENT and EMPLOYEE relation.

Query: ∏ NAME (STUDENT) ∪ ∏ NAME (EMPLOYEE)

Name
Arman
Rehan
Hamza
Ahmad
Ali
Adnan
Umar

Set Difference (-)

Set Difference as its name indicates is the difference between two relations (R-S). It is denoted by a "Hyphen"(-) and it returns all the tuples(rows) which are in relation R but not in relation S. It is also a binary operator.

Notation : R – S

- Where R is the first relation
- S is the second relation

Just like union, the set difference also comes with the exception of the same set of attributes in both relations.

Let's take an example where we would like to know the names of students who are in STUDENT Relation but not in EMPLOYEE Relation.

∏ NAME(STUDENT) - ∏ NAME(EMPLOYEE)

Output:

Name
Arman
Rehan

Cartesian product (X)

Cartesian product is denoted by the "X" symbol. Let's say we have two relations R and S. Cartesian product will combine every tuple(row) from R with all the tuples from S.

Notation: R X S

Where R is the first relation

S is the second relation

As we can see from the notation it is also a binary operator.

Let's combine the two relations STUDENT and EMPLOYEE.

Query : STUDENT X EMPLOYEE

Database Systems

Roll No	Name	Age	Employee ID	Name	Age
1	Arman	20	E1	Ali	20
1	Arman	20	E2	Hamza	23
1	Arman	20	E3	Adnan	26
1	Arman	20	E4	Umar	20
1	Arman	20	E5	Ahmad	24
2	Rehan	23	E1	Ali	20
2	Rehan	23	E2	Hamza	23
2	Rehan	23	E3	Adnan	26
2	Rehan	23	E4	Umar	20
2	Rehan	23	E5	Ahmad	24
3	Hamza	26	E1	Ali	20
3	Hamza	26	E2	Hamza	23
3	Hamza	26	E3	Adnan	26
3	Hamza	26	E4	Umar	20
3	Hamza	26	E5	Ahmad	24
4	Ahmad	20	E1	Ali	20
4	Ahmad	20	E2	Hamza	23
4	Ahmad	20	E3	Adnan	26
4	Ahmad	20	E4	Umar	20
4	Ahmad	20	E5	Ahmad	24
5	Ali	24	E1	Ali	20
5	Ali	24	E2	Hamza	23
5	Ali	24	E3	Adnan	26
5	Ali	24	E4	Umar	20
5	Ali	24	E5	Ahmad	24
6	Ahmad	22	E1	Ali	20
6	Ahmad	22	E2	Hamza	23
6	Ahmad	22	E3	Adnan	26
6	Ahmad	22	E4	Umar	20
6	Ahmad	22	E5	Ahmad	24

Rename (ρ)

Rename operation is denoted by "Rho"(ρ). As its name suggests it is used to rename the output relation. Rename operator too is a **binary operator**. **Notation: ρ(R,S)**

- Where R is the new relation name
  S is the old relation name

Let's have an example to clarify this Suppose we are fetching the names of students from STUDENT relation. We would like to rename this relation as STUDENT_NAME.

ρ(STUDENT_NAME,∏ NAME(STUDENT))

Student _Name

Name
Arman
Rehan
Hamza
Ahmad
Ali

- **Select (σ) is used to retrieve tuples(rows) based on certain conditions.**
- **Project (∏) is used to retrieve attributes(columns) from the relation.**
- **Union (∪) is used to retrieve all the tuples from two relations.**
- **Set Difference (-) is used to retrieve the tuples which are present in R but not in S(R-S).**
- **Cartesian product (X) is used to combine each tuple from the first relation with each tuple from the second relation.**
- **Rename (ρ) is used to rename the output relation.**

Derived Operations

Also known as extended operations, these operations can be derived from basic operations and hence named Derived Operations. These include three operations: Join Operations, Intersection operations, and Division operations.

Join Operations

Join Operation in DBMS are binary operations that allow us to combine two or more relations.
They are further classified into two types: Inner Join, and Outer Join.

First, let's have two relations

EMPLOYEE consisting of E_NO, E_NAME, CITY and EXPERIENCE. EMPLOYEE table contains employee's information such as id, name, city, and experience of employee (In Years). The other relation is DEPARTMENT consisting of D_NO, D_NAME, E_NO and MIN_EXPERIENCE.

DEPARTMENT table defines the mapping of an employee to their department. It contains Department Number, Department Name, Employee Id of the employee working in that department, and the minimum experience required (In Years) to be in that department.

EMPLOYEE

E_NO	E_NAME	CITY	EXPERIENCE
E-1	Ali	Faisalabad	04
E-2	Ahmad	Lahore	09
E-3	Umar	Karachi	03
E-4	Adnan	Islamabad	07

Database Systems

DEPARTMENT

D_NO	D_NAME	E_NO	MIN_EXPERIENCE
D-1	HR	E-1	03
D-2	IT	E-2	05
D-3	Marketing	E-3	02

Also, let's have the Cartesian product of the above two relations. It will be much easier to understand Join Operations.

E_NO	E_NAME	CITY	EXPERIENCE	D_NO	D_NAME	E_NO	MIN_EXPERIENCE
E-1	Ali	Faisalabad	04	D-1	HR	E-1	03
E-1	Ali	Faisalabad	04	D-2	IT	E-2	05
E-1	Ali	Faisalabad	04	D-3	Marketing	E-3	02
E-2	Ahmad	Lahore	09	D-1	HR	E-1	03
E-2	Ahmad	Lahore	09	D-2	IT	E-2	05
E-2	Ahmad	Lahore	09	D-3	Marketing	E-3	02
E-3	Umar	Karachi	03	D-1	HR	E-1	03
E-3	Umar	Karachi	03	D-2	IT	E-2	05
E-3	Umar	Karachi	03	D-3	Marketing	E-3	02
E-4	Adnan	Islamabad	07	D-1	HR	E-1	03
E-4	Adnan	Islamabad	07	D-2	IT	E-2	05
E-4	Adnan	Islamabad	07	D-3	Marketing	E-3	02

Inner Join

When perform Inner Join, only those tuples returned that satisfy the certain condition. It is also classified into three types: **Theta Join, Equi Join and Natural Join.**

Theta Join (θ)

Theta Join combines two relations using a condition. This condition is represented by the symbol "theta"(θ). Here conditions can be inequality conditions such as >,<,>=,<=, etc.

Notation : R ⋈θ S

- **Where R is the first relation**
- **S is the second relation**

Let's have a simple example to understand this.

Suppose we want a relation where EXPERIENCE from EMPLOYEE >= MIN_EXPERIENCE from DEPARTMENT.

Query:

EMPLOYEE ⋈θ EMPLOYEE.EXPERIENCE>=DEPARTMENT.MIN_EXPERIENCE DEPARTMENT

E_NO	E_NAME	CITY	EXPERIENCE	D_NO	D_NAME	E_NO	MIN_EXPERIENCE
E-1	Ali	Faisalabad	04	D-1	HR	E-1	03
E-1	Ali	Faisalabad	04	D-3	Marketing	E-3	02
E-2	Ahmad	Lahore	09	D-1	HR	E-1	03
E-2	Ahmad	Lahore	09	D-2	IT	E-2	05
E-2	Ahmad	Lahore	09	D-3	Marketing	E-3	02
E-3	Umar	Karachi	03	D-1	HR	E-1	03
E-3	Umar	Karachi	03	D-3	Marketing	E-3	02
E-4	Adnan	Islamabad	07	D-1	HR	E-1	03
E-4	Adnan	Islamabad	07	D-2	IT	E-2	05
E-4	Adnan	Islamabad	07	D-3	Marketing	E-3	02

Check the Cartesian Product, if in any tuple/row EXPERIENCE >= MIN_EXPERIENCE then insert this tuple/row in output relation.

Equi Join

Equi Join is a special case of theta join where the condition can only contain **equality(=)** comparisons.
A non-equijoin is the inverse of an equi join, which occurs when you join on a condition other than "=".

Let's have an example where we would like to join EMPLOYEE and DEPARTMENT relation where E_NO from EMPLOYEE = E_NO from DEPARTMENT.

Query:

EMPLOYEE ⋈EMPLOYEE.E_NO = DEPARTMENT.E_NO DEPARTMENT

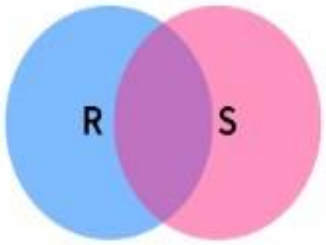
E_NO	E_NAME	CITY	EXPERIENCE	D_NO	D_NAME	E_NO	MIN_EXPERIENCE
E-1	Ali	Faisalabad	04	D-1	HR	E-1	03
E-2	Ahmad	Lahore	09	D-2	IT	E-2	05
E-3	Umar	Karachi	03	D-3	Marketing	E-3	02

But unlike the above operation, where we have two columns of E_NO, here we are having only one column of E_NO. This is because Natural Join automatically keeps a single copy of a common attribute.

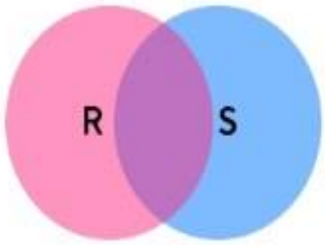
Outer Join

Unlike Inner Join which includes the tuple that satisfies the given condition, Outer Join also includes some/all the tuples which don't satisfy the given condition. It is also of three types: Left Outer Join, Right Outer Join, and Full Outer Join.

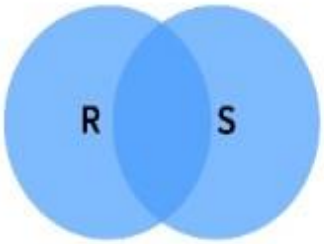
Let's say we have two relations R and S, then
Below is the representation of Left, Right, and Full Outer Joins.



Left Outer Join



Right Outer Join



Full Outer Join

Left Outer Join

As we can see from the diagram, Left Outer Join returns the matching tuples(tuples present in both relations) and the tuples which are only present in Left Relation, here R.

However, if the matching tuples are NULL, then attributes/columns of Right Relation, here S are made NULL in the output relation.

Let's understand this a bit more using an example:

Query:

EMPLOYEE ⟕EMPLOYEE.E_NO = DEPARTMENT.E_NO DEPARTMENT

Here we are combining EMPLOYEE and DEPARTMENT relation with the constraint that EMPLOYEE's E_NO must be equal to DEPARTMENT's E_NO.

E_ NO	E_ NAME	CITY	EXPERIENCE	D_ NO	D_ NAME	E_ NO	MIN_EXPERIENCE
E-1	Ali	Faisalabad	04	D-1	H R	E-1	03
E-2	Ahmad	Lahore	09	D-2	IT	E-2	05
E-3	Umar	Karachi	03	D-3	Marketing	E-3	02
E-4	Adnan	Islamabad	07	-	-	-	-

As you can see here, all the tuples from left, i.e., EMPLOYEE relation are present. But E-4 is not satisfying the given condition, i.e., E_NO from EMPLOYEE must be equal to E_NO from DEPARTMENT, still it is included in the output relation. This is because Outer Join also includes some/all the tuples which don't satisfy the condition. That's why Outer Join marked E-4's corresponding tuple/row from DEPARTMENT as NULL.

Right Outer Join

Right Outer Join **returns the matching tuples and the tuples which are only present in Right Relation** here S.

The same happens with the Right Outer Join, if the matching tuples are NULL, then the attributes of Left Relation, here R are made NULL in the output relation.

We will combine EMPLOYEE and DEPARTMENT relations with the same constraint as above.

Query:

EMPLOYEE ⋈EMPLOYEE.E_NO = DEPARTMENT.E_NO DEPARTMENT

E_ NO	E_ NAME	CITY	EXPERIENCE	D_NO	D_NAME	E_NO	MIN_ EXPERIENCE
E-1	Ali	Faisalabad	04	D-1	H R	E-1	03
E-2	Ahmad	Lahore	09	D-2	IT	E-2	05
E-3	Umar	Karachi	03	D-3	Marketing	E-3	02

As all the tuples from DEPARTMENT relation have a corresponding E_NO in EMPLOYEE relation, therefore no tuple from EMPLOYEE relation contains a NULL.

Full Outer Join

Full Outer Join **returns all the tuples from both relations**. However, if there are no matching tuples then, their respective attributes are made NULL in output relation.

Again, combine the EMPLOYEE and DEPARTMENT relation with the same constraint.

Query:

EMPLOYEE ⟗EMPLOYEE.E_NO = DEPARTMENT.E_NO DEPARTMENT

E_ NO	E_ NAME	CITY	EXPERIENCE	D_ NO	D_ NAME	E_ NO	MIN_EXPERIENCE
E-1	Ali	Faisalabad	04	D-1	H R	E-1	03
E-2	Ahmad	Lahore	09	D-2	IT	E-2	05
E-3	Umar	Karachi	03	D-3	Marketing	E-3	02
E-4	Adnan	Islamabad	07	-	-	-	-

Intersection (∩)

Intersection operation is done by Intersection Operator which is represented by "intersection"(∩).It is the same as the intersection operator from set theory, i.e., it selects all the tuples which are present in both relations. It is a **binary operator** as it requires two operands. Also, it **eliminates duplicates**.
Notation : R ∩ S

- **Where R is the first relation**
- **S is the second relation**

Let's have an example to clarify the concept:
Suppose we want the names which are present in STUDENT as well as in EMPLOYEE relation, Relations we used in Basic Operations.

Query:

∏ NAME(STUDENT) ∩ ∏ NAME(EMPLOYEE)

Name
Hamza
Ahmad
Ali

Division (÷)

Division Operation is represented by "division"(÷ or /) operator and is used in queries that involve keywords **"every"**, **"all"**, etc.
Notation : R(X,Y)/S(Y)
Here,
R is the first relation from which data is retrieved.
S is the second relation that will help to retrieve the data.
X and Y are the attributes/columns present in relation.

We can have multiple attributes in relation, but keep in mind that attributes of S must be a proper subset of attributes of R.
For each corresponding value of Y, the above notation will return us the value of X from tuple<X,Y> which exists **everywhere**.

Let's have two relations, ENROLLED and COURSE. ENROLLED consist of two attributes STUDENT_ID and COURSE_ID. It denotes the map of students who are enrolled in given courses.

COURSE contains the list of courses available.
See, here attributes/columns of COURSE relation are a proper subset of attributes/columns of ENROLLED relation. Hence Division operation can be used here.

ENROLLED

STUDENT_ID	**COURSE_ID**
Student_1	DBMS
Student_2	DBMS
Student_1	OS
Student_3	OS

COURSE

COURSE_ID
DBMS
OS

Now the query is to return the STUDENT_ID of students who are enrolled in **every** course.

Database Systems

Query:

ENROLLED(STUDENT_ID, COURSE_ID)/COURSE(COURSE_ID)

This will return the following relation as output.

STUDENT_ID

STUDENT_ID
Student_1

Database System

Structured Query Language

SQL, or Structured Query Language, is a powerful database management and manipulation tool. It allows users to effortlessly interact with databases by providing a standardized language for operations such as data retrieval and updating. SQL's ease of use and adaptability make it an essential tool for anybody working with databases.

SQL, or Structured Query Language, is the language used to interface with relational databases. It is the bonding agent that allows software applications to efficiently handle and retrieve data. SQL, in essence, allows you to interface with databases by writing queries that execute various activities such as retrieving, updating, and removing data.

Components of SQL

The components of the SQL can be likened together to create a powerful application.

- **Data Definition Language (DDL)**:

It is the architect of the database, DDL commands include CREATE, ALTER, and DROP, shaping the structure of your data.

- **Data Manipulation Language (DML)**:

Visualize DML as the construction crew; it handles the operations like SELECT, INSERT, UPDATE, and DELETE, structuring the content within your database.

- **Data Control Language (DCL)**:

Security is the priority, and DCL commands like GRANT and REVOKE control access permissions, ensuring that only authorized users enter the gates.

- **Transaction Control Language (TCL)**:

Database System

This is the directing body, TCL commands COMMIT and ROLLBACK manage the flow of transactions, ensuring data integrity.

Understanding these components enables you to use SQL with expertise, maximizing the potential to successfully organize, query, and preserve your data.

SQL Commands

SQL commands are the foundation of database management, allowing users to easily interact with and alter data. Structured Query Language (SQL) makes it easier to retrieve, update, and manage data in databases.

SELECT commands, which serve as SQL's vision, are used to retrieve data. The INSERT, UPDATE, and DELETE instructions serve as the hands, of updating data entries. JOIN commands join tables together, forming associations that improve data retrieval.

Consider SQL to be a chef in a kitchen: SELECT is for selecting ingredients, INSERT is for adding new things to the menu, UPDATE is for revising recipes, and DELETE is for deleting undesired dishes. The INDEX commands organize the kitchen for efficiency, while the GROUP BY and ORDER BY commands organize the final presentation.

The SELECT Statement

SELECT Column or Columns

FROM table or tables

WHERE condition

GROUP BY field or fields

ORDER BY Sort order

Types of SQL Databases

There are many popular RDBMS available to work with. Some of the most popular RDBMS are listed below −

Database System

- MySQL
- MS SQL Server
- ORACLE
- MS ACCESS
- PostgreSQL
- SQLite

SQL Join and Subquery

A join is a query that combines records from two or more tables. A join will be performed whenever multiple tables appear in the FROM clause of the query. The select list of the query can select any columns from any of these tables. If join condition is omitted or invalid then a Cartesian product is formed. If any two of these tables have a column name in common, then must qualify these columns throughout the query with table or table alias names to avoid ambiguity. Most join queries contain at least one join condition, either in the FROM clause or in the WHERE clause.

Advantages Of Joins:

- The advantage of a join includes that it executes faster.
- The retrieval time of the query using joins almost always will be faster than that of a subquery.
- By using joins, you can minimize the calculation burden on the database i.e., instead of multiple queries using one join query. This means you can make better use of the database's abilities to search through, filter, sort, etc.

Disadvantages Of Joins:

- Disadvantage of using joins includes that they are not as easy to read as subqueries.
- More joins in a query means the database server has to do more work, which means that it is more time consuming process to retrieve data

Subquery

A Subquery or Inner query or Nested query is a query within SQL query and embedded within the WHERE clause. A Subquery is a SELECT statement that is embedded in a clause of another SQL statement. They can be very useful to select rows from a table with a condition that depends on the data in the same or another table. A Subquery is used to return data that will be used in the main query as a condition to further restrict the data to be retrieved. The subquery can be placed in the following SQL clauses they are WHERE clause, HAVING clause, FROM clause.

Advantages Of Subquery:

- Subqueries divide the complex query into isolated parts so that a complex query can be broken down into a series of logical steps.
- It is easy to understand and code maintenance is also at ease.
- Subqueries allow you to use the results of another query in the outer query.
- In some cases, subqueries can replace complex joins and unions.

Disadvantages of Subquery:

- The optimizer is more mature for MYSQL for joins than for subqueries, so in many cases a statement that uses a subquery can be executed more efficiently if you rewrite it as join.
- We cannot modify a table and select from the same table within a subquery in the same SQL statement.

How to Group and Aggregate Data Using SQL?

Database System

There can be many columns in a database table, so sometimes it can become difficult and time taking to find the same type of data in these columns. The GROUP BY statement groups the identical rows present in the columns of a table. GROUP BY statement in conjunction with SQL aggregate functions (COUNT (), MAX(), MIN(), SUM(), AVG() etc.) help us to analyze the data efficiently.

- COUNT counts how many rows are in a particular column.
- SUM adds together all the values in a particular column.
- MIN and MAX return the lowest and highest values in a particular column, respectively.
- AVG calculates the average of a group of selected values.

Database Backup Recovery

Backup and Recovery both terms are data protection terminology. An efficient backup and recovery system is critical for any firm to secure its precious data. However, backup and recovery are distinct techniques, backup stores a copy of the complete database onto storage media. In contrast, recovery is the process of retrieving lost data from backup storage mediums.

Backup

Backup refers to the storage of a *replication* of the *original data* that may be utilized in the event of data loss. Backup is considered one of the best data security methods, and organizations should secure their important data by utilizing the backup process. Backup can be accomplished by storing a backup copy of the original data on storage devices or in a database.

The frequency of backup creation might vary depending on the importance of the data. For example, if the data is particularly valuable, it must be backed up daily. *Monthly and quarterly backups* are the same as daily backups but are only performed on the last day of the month or quarter.

Database System

Nowadays, backups are created in the cloud due to technological advancements because it offers highly feasible storage and simple management. There are numerous backup types available, including complete backup, incremental backup, local backup, mirror backup, and others.

Types of Data Backup

There are mainly three types of data backup, ***full, incremental, and differential backup***.

1. Full Backup

It is a simple and full backup procedure that copies all of your data to another media set, including a tape, disk, or CD. As a result, a complete copy of all your data is provided on a single media set. It takes a longer time to complete and takes much storage space.

2. Incremental Backup

Incremental backups take up less space and time than differential and full backups, but they are the most time-consuming technique for restoring a full system. They're ideal for backing up data that hasn't changed in a long time. However, there is no method to predict how much space you would use for future backups.

3. Differential Backup

Differential backups are a combination of executing complete backups and incremental backups on a regular basis. Differential backups are similar to incremental backups in that they store data about changes to a database or server. These backups make it simple to recover a whole copy of a database or server from a single file. These backups are ideal for swiftly restoring databases and servers without rebuilding everything.

These backups are very useful because they allow you to recover a database or server swiftly. You don't have to generate a completely new version of it. Instead,

you apply the most recent code changes, restore a differential backup, and have a working copy of the database or server.

Features of Backup

There are various features of Backup. Some of the backup features are as follows:

1. It is generally a data replica that is utilized to restore the actual data in the case of data loss/damage.
2. It makes the process of data recovery simple and easy.
3. It is commonly utilized in manufacturing environments.
4. It provides data security to the users.
5. It is a cost-effective process to retrieve the data.

What is Recovery?

A ***database recovery*** system is an essential component of a ***Database Management System*** that assures data consistency even after a system failure. The process of restoring lost data is referred to as recovery. Even if the data was backed up, it could still be recovered by employing various recovery procedures. When a database fails, there is a risk of data loss. Therefore, the recovery procedure aids in improving the database's reliability. Moreover, if any of the transactions fails in the process of some activities, data recovery becomes a critical task and the only option to save the lost data.

In this case, the failure could be any type, including system failure, concurrency control enforcement, transaction errors, exception conditions, disk failure, and disasters. Any event that results in downtime would require recovery. There are various recovery processes, including Steal/no-steal and force/no-force policies, shadowing, caching, before and after images of the data item, UNDO, REDO recovery, etc.

Features of recovery

Database System

There are various features of Backup. Some of the backup features are as follows:

1. It is a process for restoring lost, damaged, or corrupted data to its original state.
2. The process of recovering is expensive.
3. When there is a failure, it refers to recovering the lost data.
4. It increases the database's reliability.
5. It is rarely utilized in production environments.

Difference Between Backup and recovery

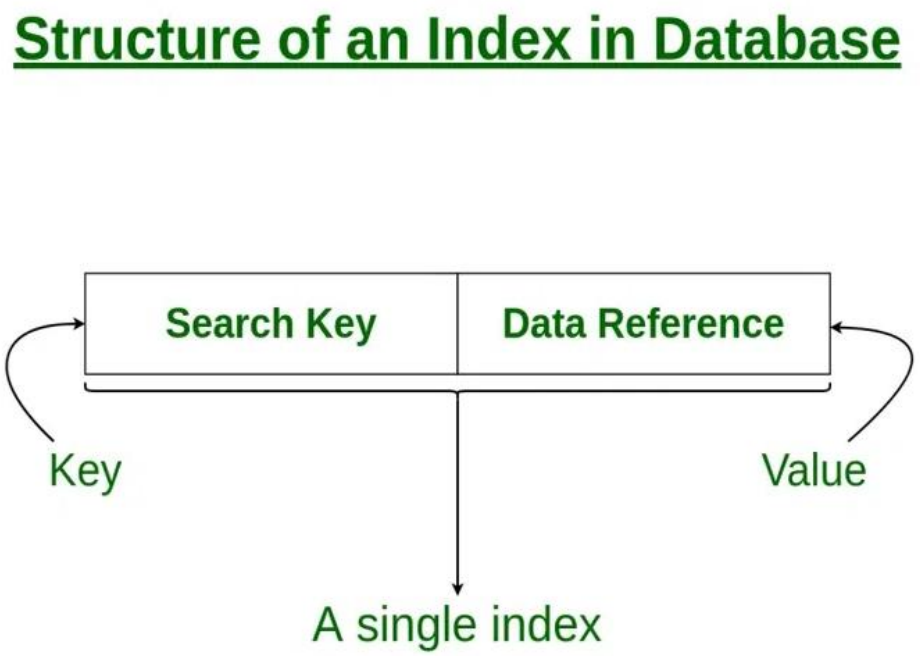
There are various key differences between ***Backup and Recovery***. Some of the key differences between the Backup and Recovery are as follows:

1. A backup is a replication of data that is utilized to recover the original data in the event of a data loss. In contrast, recovery is the process of restoring inaccessible, damaged, lost, corrupted, or formatted data to its original state.
2. A backup is a data replication. On the other hand, recovery is the process of storing the database.
3. Taking backups does not determine their duration or systematic use. On the other hand, recovery techniques are extremely beneficial. There are various recovery options, including image-based backup, continuous replication or snapshot, etc.
4. The use of backup production is quite prevalent. In contrast, recovery production is extremely rare.
5. Backup necessitates additional storage space. On the other hand, Recovery does not require additional external storage space because restoring is done internally.

Indexing in Databases

Indexing improves database performance by minimizing the number of disc visits required to fulfill a query. It is a data structure technique used to locate and quickly access data in databases. Several database fields are used to generate indexes. The main key or candidate key of the table is duplicated in

the first column, which is the Search key. To speed up data retrieval, the values are also kept in sorted order. It should be highlighted that sorting the data is not required. The second column is the Data Reference or Pointer which contains a set of pointers holding the address of the disk block where that particular key value can be found.



Attributes of Indexing

- **Access Types:** This refers to the type of access such as value-based search, range access, etc.
- **Access Time**: It refers to the time needed to find a particular data element or set of elements.
- **Insertion Time:** It refers to the time taken to find the appropriate space and insert new data.
- **Deletion Time:** Time taken to find an item and delete it as well as update the index structure.
- **Space Overhead:** It refers to the additional space required by the index.

Database System

NoSQL System

NoSQL is a type of database management system (DBMS) that is designed to handle and store large volumes of unstructured and semi-structured data. Unlike traditional relational databases that use tables with pre-defined schemas to store data, NoSQL databases use flexible data models that can adapt to changes in data structures and are capable of scaling horizontally to handle growing amounts of data.

The term NoSQL originally referred to “non-SQL” or “non-relational” databases, but the term has since evolved to mean “not only SQL,” as NoSQL databases have expanded to include a wide range of different database architectures and data models.

NoSQL databases are generally classified into four main categories:

1. Document databases: These databases store data as semi-structured documents, such as JSON or XML, and can be queried using document-oriented query languages.
2. Key-value stores: These databases store data as key-value pairs, and are optimized for simple and fast read/write operations.
3. Column-family stores: These databases store data as column families, which are sets of columns that are treated as a single entity. They are optimized for fast and efficient querying of large amounts of data.
4. Graph databases: These databases store data as nodes and edges, and are designed to handle complex relationships between data.

NoSQL databases are often used in applications where there is a high volume of data that needs to be processed and analyzed in real-time, such as social media analytics, e-commerce, and gaming. They can also be used for other applications, such as content management systems, document management, and customer relationship management.

Database System

However, NoSQL databases may not be suitable for all applications, as they may not provide the same level of data consistency and transactional guarantees as traditional relational databases. It is important to carefully evaluate the specific needs of an application when choosing a database management system.

Key Features of NoSQL:

1. **Dynamic schema:** NoSQL databases do not have a fixed schema and can accommodate changing data structures without the need for migrations or schema alterations.
2. **Horizontal scalability:** NoSQL databases are designed to scale out by adding more nodes to a database cluster, making them well-suited for handling large amounts of data and high levels of traffic.
3. **Document-based:** Some NoSQL databases, such as MongoDB, use a document-based data model, where data is stored in a **scales**semi-structured format, such as JSON or BSON.
4. **Key-value-based:** Other NoSQL databases, such as Redis, use a key-value data model, where data is stored as a collection of key-value pairs.
5. **Column-based:** Some NoSQL databases, such as Cassandra, use a column-based data model, where data is organized into columns instead of rows.
6. **Distributed and high availability:** NoSQL databases are often designed to be highly available and to automatically handle node failures and data replication across multiple nodes in a database cluster.
7. **Flexibility:** NoSQL databases allow developers to store and retrieve data in a flexible and dynamic manner, with support for multiple data types and changing data structures.
8. **Performance:** NoSQL databases are optimized for high performance and can handle a high volume of reads and writes, making them suitable for big data and real-time applications.

Database System